ДЭВИД ВЭЙЛ,
МАРТИН О`ХЭНЛОН

PYTHON 3

MINECRAFT

ПРОГРАММИРУЙ СВОЙ МИР

ПИТЕР®

Adventures in Minecraft®
Second Edition

Martin O'Hanlon and David Whale

WILEY

ДЭВИД ВЭЙЛ,
МАРТИН О`ХЭНЛОН

PYTHON 3

MINECRAFT

ПРОГРАММИРУЙ СВОЙ МИР

ПИТЕР®
Санкт-Петербург · Москва · Екатеринбург · Воронеж
Нижний Новгород · Ростов-на-Дону
Самара · Минск
2018

Дэвид Вэйл, Мартин О`Хэнлон

Minecraft. Программируй свой мир на Python

2-е международное издание

Серия «Вы и ваш ребенок»

ББК 32.973.23-018.2
УДК 004.9

Дэвид Вэйл, Мартин О`Хэнлон

В97 Minecraft. Программируй свой мир на Python. 2-е межд. изд. — СПб.: Питер, 2018. — 224 с.: ил. — (Серия «Вы и ваш ребенок»).

ISBN 978-5-4461-0951-7

Любишь играть в Minecraft? Тебе нравится узнавать новое и придумывать то, чего раньше не существовало? Хочешь построить собственный виртуальный мир, которому будут завидовать все друзья?

Можно ли объединить Minecraft и программирование? Нужно!

Теперь ты будешь не только играть и жить в удивительном мире Minecraft, но и научишься программировать на Python. Простые инструкции и советы помогут воплотить свои идеи в жизнь, построить дом и 3D-копировальную машину, найти сокровища и даже завести в своем «огороде» гигантские работающие часы.

Прочитай эту книгу и превратись в настоящего демиурга, который способен создать свой мир и защитить его от инопланетян.

Теперь на Python 3!

6+ (В соответствии с Федеральным законом от 29 декабря 2010 г. № 436-ФЗ.)

Перевод с английского А. Киселева

Заведующая редакцией *Ю. Сергиенко*

Ведущий редактор *К. Тульцева*

Художественный редактор *Д. Семенова*

Корректоры *Н. Викторова, И. Тимофеева*

Верстка *С. Романов*

ISBN 978-1119439585 англ.
ISBN 978-5-4461-0951-7

Изготовлено в России. Изготовитель: ООО «Прогресс книга». Место нахождения и фактический адрес: 194044, Россия, г. Санкт-Петербург, Б. Сампсониевский пр., д. 29А, пом. 52. Тел.: +78127037373.

Дата изготовления: 06.2018. Наименование: детская литература. Срок годности: не ограничен.

Импортер в Беларусь: ООО «ПИТЕР М», 220020, РБ, г. Минск, ул. Тимирязева, д. 121/3, к. 214, тел./факс: 208 80 01.

Налоговая льгота — общероссийский классификатор продукции ОК 034-2014, 58.11.12.000 — Книги печатные профессиональные, технические и научные.

Подписано в печать 20.06.18. Формат 84×108/16. Бумага офсетная. Усл. п. л. 23,520. Тираж 2200. Заказ № ВЗК-02785-18.

Отпечатано в АО «Первая Образцовая типография», филиал «Дом печати — ВЯТКА» в полном соответствии с качеством предоставленных материалов. 610033, г. Киров, ул. Московская, 122. Факс: (8332) 53-53-80, 62-10-36
http://www.gipp.kirov.ru; e-mail: order@gipp.kirov.ru

Оглавление

Моей жене Леони.
Без тебя не было бы этой книги.

— Мартин

Моим родителям Джан и Альфу,
показавшим мне всю ценность обучения.

— Дэвид

Об авторах

Мартин О'Ханлон (Martin O'Hanlon) всю свою сознательную жизнь занимался проектированием и программированием компьютерных систем. Страсть к программированию и стремление помочь окружающим в обучении привели его к созданию блога (www.stuffaboutcode.com), где он делится опытом, навыками и идеями. Мартин регулярно проводит презентации и семинары по программированию в Minecraft для программистов, преподавателей и молодежи с целью вдохновить их попробовать что-то новенькое и превратить программирование в забаву.

Дэвид Вейл (David Whale) провел большую часть своей жизни за написанием программ для таких компьютерных устройств, которые вам даже сложно представить. Он вкусил горечь встречи с программными ошибками в 11-летнем возрасте, когда учился в школе, и с тех пор занимается программированием и помогает другим его изучать. Работает с образовательным фондом Micro:bit Educational Foundation над доступностью компьютерного программирования для всех, а также регулярно на общественных началах выступает в Институте инженерии и технологии (IET), помогает в школах, в выходные ведет занятия в компьютерных клубах, участвует в судействе школьных конкурсов и организует семинары по программированию для молодежи на общественных мероприятиях по всей Великобритании. Следить за его приключениями вы можете в Twitter: www.twitter.com/whaleygeek.

Благодарности

В создании этой книги приняло участие множество людей. Их слишком много, чтобы здесь можно было перечислить всех. Тем не менее мы оба хотим выразить особую благодарность:

- сотрудникам компании Mojang за создание замечательной игры, отдать должное их гению и предусмотрительности, благодаря которой они сделали игру программируемой. Без этой предусмотрительности данная книга никогда не увидела бы свет;
- нашим испытателям и юным экспертам Minecraft Захарию Игельману (Zachary Igielman), Лорену Трасслеру (Lauren Trussler), Сэму Вейлу (Sam Whale), Бену Фодену (Ben Foden) и Риа Паришу (Ria Parish), опробовавшим наши программы и представившим очень полезные отзывы, без которых мы никогда не узнали бы, насколько понятно изложены наши мысли для целевой возрастной аудитории;
- Саре Райт (Sarah Wright) за превосходные иллюстрации к книге. Созданные ею замечательные образцы художественного искусства остроумно и наглядно представляют понятия, описываемые в каждом приключении;
- Бену Рамачандре (Ben Ramachandra), юноше, с которым мы познакомились на рождественских мероприятиях Fire Tech Camp 2013 года в Имперском колледже (Imperial College) Лондона. Он был так решительно настроен следовать курсом использования Python в Minecraft, что в какой-то момент нас посетила идея, ставшая искрой, которая зажгла пламя желания написать эту книгу;
- и конечно, мы очень благодарны Керри-Энн Филбин (Carrie-Anne Philbin) за ее проницательность и решимость, благодаря которым она написала свою первую книгу «Adventures in Raspberry Pi», без которой не появилась бы на свет ее серия книг «Adventures». Теперь вы понимаете, что все началось из-за вас, Кэрри-Энн?

От издательства

Ваши замечания, предложения, вопросы отправляйте по адресу comp@piter.com (издательство «Питер», компьютерная редакция).

Мы будем рады узнать ваше мнение!

На веб-сайте издательства www.piter.com вы найдете подробную информацию о наших книгах.

Предисловие

Я учитель, и мой главный интерес — обучение и вовлечение в этот процесс учеников. Обучение программированию с помощью Minecraft (изучение языка Python происходит на событиях Raspberry Jam и в школьном классе) вызывает у учеников больший энтузиазм, чем если бы я использовал другие методы обучения. И когда учебный план вместо этого переносит нас к теоретическим областям или другим задачам программирования, ученики регулярно спрашивают: «Когда мы снова будем кодить в Minecraft, сэр?» Проще говоря, Minecraft хорошо воспринимается детским мозгом, а методы программирования воспринимаются более глубоко.

Minecraft очень сильно привлекает учеников, раньше я не сталкивался с чем-то подобным. Простота этой игры обращена ко всем детям, и это понятно им так же, как и игра со строительными кубиками. И когда дети расширяют свои навыки с помощью Minecraft в качестве средства обучения — в классе, библиотеке или в специализированных технических кружках, такое обучение по-прежнему остается привлекательным. Поскольку ученики вовлечены в приобретение новых навыков и заинтересованы этим, использование Minecraft на школьных занятиях очень полезно как для преподавателя, так и для учеников.

Я считаю, что одна из причин, по которой игра Minecraft эффективна в качестве инструмента обучения, — это немедленная визуальная обратная связь, будь то в форме построенного дома, трехмерных черепах, целого города или трекера ISS. В конце концов, Minecraft — это песочница, а это значит, что у Minecraft нет ограничений на то, что вы можете сделать, — даже на бесплатной версии Raspberry Pi.

Ученики помнят, что делают в Minecraft, и это служит отличным справочным материалом — помогает выделять ключевые понятия и становится хорошим подспорьем для обучения. Например, я регулярно ссылаюсь на двумерные массивы, напоминая учащимся урок, на котором мы создали стену размером 32 × 32 из случайных блоков.

В моей школе в Великобритании, в Северном Уорикшире, есть 10 экземпляров книги «*Minecraft. Программируй свой мир на Python*». Мы используем их, чтобы помочь ученикам самостоятельно работать над дальнейшими проектами после знакомства с основами. Например, мы можем вместе выполнить задание, такое как создание радуги, когда идет персонаж, а затем ученики могут перейти к самостоятельному использованию книги, чтобы построить дом или сделать мини-игру. Прелесть книги заключается в том, что Приключения в ней могут изучаться по частям или целиком. Вы можете менять их и работать в них в разных направлениях. Я обнаружил, что большинство создаваемых мной ресурсов начинаются с идей или с кода, который я черпаю из книги Дэвида и Мартина. Эта книга вдохновляет людей всех возрастов на знакомство с программированием. По сути, она представляет собой базу знаний для учителей, родителей и, самое главное, для детей.

Было бы наивно полагать, что обучение ограничивается лишь школьным классом. Такие мероприятия, как Code Club, CoderDojo и Raspberry Jam, привлекают людей во всем мире

к программированию в Minecraft. Учиться программровать можно как дома с семьей, так и самостоятельно. Эта книга актуальна в каждом таком случае.

Я как-то услышал, что Мартин описал эту книгу как «незаметное обучение». Я думаю, что это и есть ее суть: вы делаете классные вещи с кодом и в процессе узнаете, что такое итерации, используете обработку файлов, условные конструкции и делаете в итоге гораздо больше. Ученики и дети развлекаются, постоянно совревнуются, участвуют в решении проблем и паралелльно учатся программированию.

Желаю вам удачи в приключениях в Minecraft!

— Крис Пенн (Chris Penn), учитель информатики и computer science, школа Nicholas Chamberlaine, Уорикшир, Великобритания

Этим читателям тоже есть что сказать о книге «*Minecraft. Программируй свой мир на Python*»:

«Книга всегда спасает меня в работе! Использую ее каждый день, Давид и Мартин предоставили замечательные ресурсы для занятий!»

— Боб (Bob), преподаватель, США

«Мы используем главы из книги в нашей учебной программе «Computer science для 8 лет», а также регулярно пользуемся материалами книги в Raspberry Jam».

— Джон (Jon), преподаватель и организатор Raspberry Jam, Халл, Великобритания

«Это ПОТРЯСАЮЩАЯ книга, наполненная захватывающими способами для знакомства с принципами Python и системой автоматизированного трехмерного проектирования».

— Фрэнк (Frank), организатор Raspberry Jam, Лондон, Великобритания

«Я рекомендую эту книгу родителям на каждом мероприятии, куда я хожу, даже в Китае. Здесь началось мое путешествие по миру программирования!»

— Джемма (Gemma), арт-деятель, Манчестер, Великобритания

«Книга отлично подходит для индивидуальных и командных соревнований в Raspberry Jam, включая семейные мероприятия. Мы были вдохновлены указателями на большее количество проектов».

— Клэр (Claire), преподаватель и организатор Raspberry Jam, Лидс, Великобритания

«Бесценный ресурс. Детям Блэкпула очень нравится Raspberry Jam!»

— Лес (Les), методист и организатор Raspberry Jam, Блэкпул, Великобритания

Введение

Вы любите приключения? Вам нравится заниматься исследованиями и получать новые навыки? Вы большой поклонник игры Minecraft и хотите расширить свои возможности, научившись писать компьютерные программы, взаимодействующие с игрой, чтобы вызывать восхищение у друзей? Если вы отвечаете «да», эта книга для вас.

Что такое Minecraft

Minecraft — это инди-игра в жанре песочницы, где можно строить сооружения, собирать предметы, добывать минералы и сражаться с монстрами. Она выглядит как трехмерный виртуальный мир, состоящий из разных кубических блоков; каждый блок занимает свое место в трехмерной решетке мира. На рис. 1 показан пример ландшафта в мире Minecraft.

Виртуальный мир

В играх, относящихся к жанру песочницы, игрок находится внутри виртуального мира (как в обычной детской песочнице, но очень большой). Вместо прохождения определенных уровней в заданном порядке здесь вы путешествуете по виртуальному миру и принимаете решения, исходя из своих целей и способов их достижения. Поскольку выбор приходится делать в самом начале, игры в жанре песочницы обладают почти неограниченными возможностями. Вы создаете собственные сюжеты и перемещаетесь в трехмерном мире, приобретаете новые навыки и возможности, обнаруживая их случайно или в ходе экспериментов.

Персонаж в Minecraft, воплощение игрока — его аватар называется Стив (Steve). Игрок управляет Стивом в виртуальном мире, стремясь достичь поставленной цели. Если удастся благополучно пережить первую ночь, когда одолевают монстры, вы сможете следовать своим целям, взаимодействовать с другими игроками и строить огромные сооружения, ограничиваясь лишь собственной фантазией.

Игры в жанре песочницы позволяют игроку самому решать, как вести игру, не вынуждая его двигаться маршрутами, заложенными создателями игры. Подробнее прочесть об играх этого жанра можно по адресу: http://en.wikipedia.org/wiki/Open_world[1]. Кому-то покажется странным, что персонаж носит имя Стив (Steve). Почему был сделан именно такой выбор, вы узнаете по адресу: http://minecraft.gamepedia.com/The_Player[2].

---

[1] https://ru.wikipedia.org/wiki/Открытый_мир_(компьютерные_игры). — *Примеч. пер.*

[2] https://minecraft-ru.gamepedia.com/Игрок — *Примеч. пер.*

Рис. 1. Мир Minecraft

Как появилась игра Minecraft

Инди-игры[1] (indie games, от *англ.* independent video games — независимые видеоигры) создают конкретные люди или небольшие коллективы. Часто они работают без финансовой поддержки издателей видеоигр. Как следствие независимой природы, инди-игры часто получаются более новаторскими, чем другие, главенствующие на рынке. Согласно «Википедии», игру Minecraft создал шведский программист Маркус Перссон (Markus Persson), известный под игровым псевдонимом Notch (Нотч). Пробную версию Minecraft он продемонстрировал в 2009 году, а первый официальный релиз игры состоялся в 2011 году. Нотч основал в Швеции компанию Mojang AB, которая разработала игры Minecraft для разных компьютерных платформ, включая PC, Mac, Raspberry Pi, Linux, iOS, Android, Xbox 360, Playstation и Wii.

С захватывающей историей развития Minecraft можно познакомиться в документальном фильме «*Minecraft: The Story of Mojang*» (http://en.wikipedia.org/wiki/Minecraft:_The_Story_of_Mojang).

[1] https://ru.wikipedia.org/wiki/Инди-игра. — *Примеч. пер.*

О программировании в Minecraft

В этой книге рассказывается о программировании: игра Minecraft использована в качестве способа обучения программированию. Если вы ищете советы, как строить разные сооружения и выигрывать сражения, лучше обратиться к другим книгам по теме.

Программируя в Minecraft, можно сделать игру более захватывающей, творческой и индивидуальной. Играя в обычную игру, игроки следуют основным правилам Minecraft, заложенным создателями. А добавляя свои программы, взаимодействующие с игровым миром Minecraft, можно автоматизировать решение сложных и повторяющихся задач, таких как строительство протяженных улиц с домами и гигантских строений. Можно придать игре и ее объектам новые черты поведения, добавить новые элементы, до которых не додумались авторы. Но самое главное, что так вы приобретаете универсальный навык — учитесь программировать на языке Python, который можно применять более широко, не только для программирования в Minecraft. На рис. 2 изображена длинная улица со зданиями, построенная с помощью коротенькой программы на Python.

Рис. 2. Длинная улица со зданиями, построенная 20-строчной программой на Python

В видеоролике, объясняющем, зачем детям учиться программированию (www.youtube.com/watch?v=nKlu9yen5nc), Уильям Адамс, более известный под псевдонимом Will.i.am, сказал: «Хорошие программисты сегодня — настоящие рок-звезды». Новые навыки, полученные во время Приключений, которые описаны в этой книге, помогут вам сделать игру в Minecraft более личной, творческой и масштабной. Владение магией программирования поможет поразить друзей и партнеров по игре, заставит их задавать вопросы, как вам удается достигать невероятных результатов. Ответ прост: волшебство скрыто в программировании.

Кому адресована эта книга

Книга «*Minecraft. Программируй свой мир*» адресована мальчикам и девочкам, которые любят играть в Minecraft и хотят научиться программированию, чтобы с помощью новых знаний добиться большего. В целом книги серии «*Minecraft....*» адресованы читателям в возрасте 11–15 лет, но некоторые приключения в финале могут быть интересны и читателям более старшего возраста. Кроме того, первые главы книги испытали совсем юные любители Minecraft в возрасте от восьми лет.

Возможно, вы уже накопили серьезный игровой опыт, но испытываете досаду от того, сколько времени приходится тратить на создание новых сооружений. Или ищете способы расширить возможности игры дополнительными логическими функциями и функциями автоматизации. Какими бы ни были ваши цели, эта книга станет путеводителем в путешествии по программированию для Minecraft, а каждый искатель приключений знает, что путеводитель — самый ценный груз в рюкзаке. Свой поход вы начнете с простого, например с отправки сообщений в чат Minecraft, затем познакомитесь с основами программирования для Minecraft на языке Python и, наконец, узнаете, как использовать новые навыки для создания собственных игр внутри Minecraft. К концу путешествия вы получите знания и навыки, достаточные, чтобы стать первопроходцем в программировании для Minecraft!

Что вы узнаете

Вы познакомитесь с множеством сторон игры Minecraft и способами влияния на ее поведение с помощью программ на языке Python. Откроете тайну адресации блоков в трехмерном мире с использованием координат. Узнаете, как определять местоположение персонажа, создавать и удалять блоки в Minecraft и как определить, какой блок выбран игроком.

Вы узнаете, как писать программы на языке Python: от самых простых, таких как «Привет, мир Minecraft!», до сложных, взаимодействующих с огромными трехмерными объектами, которые, благодаря новым навыкам программирования на Python, легко штамповать. Вы также узнаете, как настроить и запустить собственный локальный сервер Minecraft на ПК.

Используя бесплатный модуль `MinecraftStuff`, входящий в состав библиотеки на языке Python, вы сможете создавать двух- и трехмерные объекты из блоков, рисовать линии и многоугольники, а также выводить текст.

Игра Minecraft имеет два основных режима: «Выживание» (Survival) и «Творчество» (Creative). На протяжении всей книги мы будем играть в режиме «Творчество» и не будем рассматривать режим «Выживание» (не очень радует, когда вы тестируете свою программу, а вас убивает враждебный моб, к тому же существует много хороших книг, описывающих, как выжить ночью в Minecraft). Однако все программы, написанные в режиме «Творчество», работают и в режиме «Выживание».

Что вы должны знать

Так как книга рассказывает о программировании в Minecraft, основное внимание будет уделено именно вопросам программирования. Мы предполагаем, что читатель уже:

1) имеет компьютер (с установленной системой Microsoft Windows), отвечающий минимальным требованиям игры Minecraft, настроенный и работающий;
2) владеет основными навыками работы с компьютером: умеет пользоваться клавиатурой и мышью; знает, как с помощью системного меню запускать программы; умеет пользоваться меню приложений, такими как File ▸ New ▸ Save (Файл ▸ Создать ▸ Сохранить);
3) имеет доступ в интернет и знает, как с помощью веб-браузера загружать из него файлы;
4) обладает идентификатором пользователя Minecraft и установленной действующей копией игры;
5) знает, как играть в Minecraft: как запускать игру, перемещаться в виртуальном мире, выбирать предметы из инвентаря, создавать и удалять блоки.

Поскольку книга рассказывает о программировании в Minecraft, мы исходим из того, что читатель имеет общее представление о том, как писать программы. В процессе путешествия мы расскажем все, что необходимо знать для обучения программированию.

Что потребуется для создания проектов

Когда писалась эта книга, предполагалось, что читатели будут работать в операционной системе Microsoft Windows. Игра Minecraft может запускаться и на других платформах, таких как Raspberry Pi под управлением Raspbian, и на персональных компьютерах под управлением Mac OS X, разновидностей Linux. Однако мы не будем рассматривать настройку этих платформ.

Чтобы упростить настройку разных компонентов, мы подготовили начальный набор инструментов, а в первом Приключении дадим пошаговые инструкции, подробно описывающие, как загрузить, установить и настроить то, что нужно. В подготовленный набор инструментов входит все необходимое, кроме самой игры Minecraft. Благодаря этому вы быстро преодолеете подготовительный этап!

Чтобы загрузить начальный набор инструментов, вам потребуется подключение к интернету. В набор включено почти все, что нужно для путешествия. Некоторые Приключения имеют особые требования, и мы будем оговаривать их в начале, чтобы вы могли подготовиться.

И больше всего во время путешествия вам понадобятся энтузиазм и желание играть в Minecraft, а также доля любопытства и готовность экспериментировать, расширять границы своих знаний.

Примечание для родителей и учителей

Мы разделили книгу на несколько отдельных приключений, которые можно считать самостоятельными проектами. Каждый из них посвящен одной особенности программирования для Minecraft. Знакомство с языком Python будет протекать постепенно, от простого к сложному. Первые приключения ориентированы исключительно на новичков, а последние охватывают более сложные разделы языка Python и заставляют читателя напрягать свои способности.

В каждом Приключении представлен отдельный практический проект с пошаговыми инструкциями (в которых читатели смогут отмечать выполненные пункты галочками), составленными в стиле, напоминающем стиль комментариев в листингах программ. Подробные описания даны во врезках «Углубляемся в код», которые ученики смогут прочесть позднее, чтобы не отвлекаться от ввода и опробования программ.

Каждое Приключение может потребовать больше чем одного урока, но все они поделены на разделы, которые можно использовать в качестве тем для отдельных уроков или занятий.

Синтаксис языка Python предусматривает оформление листингов программ с отступами слева, для выделения структуры кода — он чувствителен к регистру символов. Юным читателям могут пригодиться советы взрослых по правильному оформлению отступов и регистров символов, которые помогут им избежать ошибок при наборе текстов программ. Все программы можно загрузить со вспомогательного веб-сайта. Поэтому если вдруг в программе всплывут какие-то проблемы, сравните свою версию с нашей и посмотрите, где допустили ошибку.

Изменения, внесенные во второе издание

Во втором издании мы сохранили содержание и тот же легкий стиль повествования и внесли некоторые незначительные улучшения и исправления:

- Загружаемые начальные наборы были немного упрощены. Все координаты теперь последовательно сообщаются программе Python как абсолютные, то есть реальные координаты внутри Minecraft (а не относительно точки возрождения, как это было раньше). Это делает математику, связанную с размещением объектов в Minecraft, гораздо более понятной для детей. И за счет этого иногда выводятся большие числа. Координаты на экране теперь соотвествуют координатам, сообщенным программой на Python, которую вы пишете.
- Чтобы книга была современной и соответствовала тому, что используется в школах, все программы теперь написаны в последней версии Python 3, которую мы рекомендуем загрузить нашим читателям в Приключении 1. Единственное отличие, которое можно заметить в наших программах, это использование функции `input()` вместо `raw_ input()` (в этом плане Python 3 работает несколько иначе, чем Python 2).

- Бесплатный модуль `MinecraftStuff` Мартина был обновлен с целью упрощения некоторых функций, и это сократило количество ввода кода, необходимое для входа в программы в Приключениях 6 и 7, а также позволило включить новую функцию: Minecraft Turtle! Это полностью контролируемый блок в Minecraft, который может перемещаться в трех измерениях и позволяет читателям очень просто и быстро рисовать сложные фигуры. Он также хорошо сочетается с учебными программами, которые регулярно внедряют концепции программирования с использованием программируемой черепахи[1].

О структуре книги

Каждая глава в книге — отдельное приключение, обучающее новым навыкам и понятиям в процессе программирования и тестирования проектов. Книга организована так, что каждому приключению соответствует самостоятельный проект. Хотя, возможно, кому-то будет проще осваивать их по порядку и знакомиться с новыми понятиями программирования постепенно.

Очень важно, чтобы вы прочли Приключение 1, прежде чем делать что-то еще. Там описано, как загрузить и установить всё, что вам понадобится, и убедиться, что всё работает должным образом. В этом приключении мы перечислим главные шаги, которые придется выполнять в следующих приключениях, и будем напоминать о них на протяжении первых приключений.

Первые три приключения написаны для тех, кто мало знает или вообще ничего не знает о программировании. Мы объясним все термины и понятия по мере встречи с ними. В Приключениях 2, 3 и 4 рассмотрены ключевые элементы любой хорошей игры Minecraft. В их числе: *определение* событий, происходящих в мире Minecraft; выполнение простых *вычислений* и создание программ, *реагирующих* по-разному, например отправляющих сообщения в чат или автоматически создающих блоки. Эти три ключевых понятия — *определение*, *вычисление* и *реагирование* — мы будем использовать на протяжении всей книги для создания больших и захватывающих программ Minecraft.

Приключения 5 и 6 основываются на том, что вы узнали в предыдущих приключениях, и содержат более крупные программы, которые разрабатываются и тестируются поэтапно. В Приключении 5 рассматриваются способы, которые позволяют вносить большие объемы данных из файлов данных, чтобы сохранять и дублировать большие структуры с помощью трехмерной «копировальной машины».

Приключения 6 и 7 знакомят с модулем `MinecraftStuff`, позволяющим с помощью блоков создавать линии, окружности и другие двумерные фигуры, а также некоторые фантастические трехмерные сферы и пирамиды. Они могут стать основой для строительства гигантских сооружений, создать которые вручную очень сложно. В Приключении 7 показано, как наделить движущиеся объекты индивидуальными особенностями, чтобы сделать их движения более-менее разумными. Освоив эти приемы, вы сможете писать захватывающие «игры внутри игры», которые наверняка поразят ваших друзей.

---

[1] https://ru.wikipedia.org/wiki/Черепашья_графика — *Примеч. пер.*

Приключение 9 опирается на все концепции и навыки программирования из более ранних приключений. Вы создадите последний крупный проект — потрясающую игру со счетчиком очков и движущимися объектами, которых нужно избегать или которые можно брать с собой.

В Приложение А мы включили всестороннее справочное руководство по программным средствам, используемым в книге, таблицы с программными инструкциями, которые используются в Minecraft, а также таблицу с типами блоков, которые вы можете строить. Все это вы найдете в справочном разделе, который поможет вам в собственных проектах и разработках!

Вспомогательный веб-сайт

По всей книге вы встретите ссылки на веб-сайт *Adventures in Minecraft* (www.wiley.com/go/adventuresinminecraft2e). На нем вы найдете стартовые наборы, которые понадобятся, чтобы начать программировать в Minecraft, а также сборник видеоуроков, который мы собрали, чтобы помочь вам, если вы где-то застряли. Вы также найдете там файлы с кодом некоторых крупных проектов.

Приложение А представляет собой удобный справочный материал. Держите его при себе, когда пишете код для приключений в Minecraft. Вы также можете использовать материал из Приложения в любых других проектах, к которым приступите в будущем.

Другие источники вспомогательной информации

Компьютеры — сложные устройства, а операционные системы и программное обеспечение меняются постоянно. Мы постарались защитить вас и ваши приключения от будущих изменений, насколько это возможно, создав загружаемый начальный набор инструментов, о котором рассказано в Приключении 1 и который содержит все, что необходимо. Однако если вы столкнетесь с проблемой или вам понадобится конкретная помощь, ниже даны полезные ссылки.

Получить идентификатор пользователя и загрузить дистрибутив Minecraft можно по адресу: http://minecraft.net.

Описание игры Minecraft и ее правила: http://Minecraft-ru.gamepedia.com/Minecraft_Wiki.

Справочная информация о Microsoft Windows: http://support.microsoft.com.

Справочная информация о языке Python: www.python.org[1].

Справочная информация о среде программирования IDLE: https://docs.python.org/3/library/idle.html[2].

Сервер Minecraft: www.spigotmc.org/

---

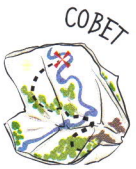
[1] На русском языке: http://python-lab.ru/. — *Примеч. пер.*

[2] На русском языке: http://www.russianlutheran.org/python/idle_doc/idle_doc.html. — *Примеч. пер.*

Условные обозначения

В книге вам постоянно будут встречаться врезки с дополнительными инструкциями и справочной информацией. Вот как они выглядят:

В этих врезках разъясняются понятия и термины, которые могут быть вам незнакомы.



В этих врезках вы найдете подсказки, упрощающие программирование.

В этих врезках содержатся важные замечания и предупреждения, цель которых — обезопасить вас и ваш компьютер на завершающих этапах разработки проектов.

В этих врезках вам будут предложены контрольные вопросы, чтобы проверить ваши знания и заставить подумать об обсуждаемой теме.

В этих врезках мы будем давать дополнительные пояснения или информацию, которая может пригодиться.

В этих врезках будут даны ссылки на видеоуроки, которые помогут вам решить поставленную задачу.

Кроме названных, в книге встречаются еще два типа врезок. Врезки с заголовком «ЗАДАНИЕ» содержат дополнительные задания, которые вы можете выполнить при желании улучшить проект, для чего может потребоваться внести изменения или добавить новые особенности. Врезки с заголовком «УГЛУБЛЯЕМСЯ В КОД» подробно описывают некоторые понятия и особенности программ для лучшего понимания языка программирования Python. Эти врезки помогут вам писать работающие программы и понять, как они действуют, чтобы увидеть дальнейшие пути улучшения.

Следуя примерам и выполняя пошаговые инструкции, вы будете вводить программный код в точности как описано в инструкциях. Пробелы в началах строк (отступы) имеют большое значение для языка Python, поэтому уделяйте особое внимание их количеству! Мы специально оформили листинги на цветном фоне, чтобы вы видели отступы в строках. Но пока не стоит волноваться по этому поводу — мы еще не раз затронем тему отступов в первых приключениях.

Иногда вам потребуется вводить очень длинные строки — длиннее, чем умещается по ширине книжной страницы. Если вы увидите значок ↩ в конце строки в листинге программы, это значит, что строка продолжается ниже и две строки — до и после значка ↩ — образуют одну строку кода; вы должны вводить их в одной строке на экране. Например, следующий код должен вводиться в одну строку, а не в две:

```
print("Welcome to Adventures in Minecraft by ↩
   Martin O'Hanlon and David Whale")
```

Если вы читаете электронную версию книги, перед вводом программ убедитесь, что устройство чтения отображает листинги без искажений: выберите как можно более мелкий шрифт, чтобы строки программ не переносились устройством при отображении. Это поможет избежать ошибок при вводе.

В конце многих приключений дана «Краткая справочная таблица» с кратким повторным описанием инструкций или понятий. Вы можете обращаться к ним, чтобы освежить в памяти. В Приложении А также есть справочный раздел, в котором показаны наиболее важные программные инструкции для Minecraft и Python. Мы надеемся, что эти таблицы будут полезны по мере продвижения вперед.

В конце каждого приключения в копилку ваших достижений будет добавляться новый значок. Получить файлы с изображениями значков, чтобы похвастаться своими достижениями друзьям, можно на веб-сайте книги (www.wiley.com/go/adventuresinminecraft2e).

Вы также можете связаться с авторами и получить помощь от других читателей на форуме *Adventures in Minecraft* по адресу www.stuffaboutcode.com/p/adventures-in-minecraft.html.

Время начинать приключения!

Приключение 1
Привет, мир Minecraft!

ЭТА КНИГА поможет вам писать программы, взаимодействующие с миром Minecraft, что откроет перед вами удивительные возможности. Писать программы вы будете на языке **Python**. Впервые возможность управления миром Minecraft из программ на Python появилась в версии Minecraft для платформы Raspberry Pi. Позднее такую поддержку добавили в Minecraft для Windows и Apple Mac. Правда, пользователям этой платформы нужно выполнить дополнительные подготовительные операции, прежде чем они смогут приступить к экспериментам. Далее мы о них расскажем.

Python — язык программирования, используемый в этой книге.

Эта книга наполнена приключениями, в ходе которых вы научитесь писать программы для игры Minecraft. Здесь описаны все доступные возможности программирования Minecraft, которые позволят вам удивить друзей и сделать игру более увлекательной. Вы откроете для себя некоторые эффектные способы перемещения персонажа, научитесь с легкостью строить в Minecraft целые города и создавать невиданные ранее сооружения.

Вместе с интерпретатором языка Python поставляется редактор программного кода IDLE, который вы будете использовать для создания, правки и запуска программ в приключениях.

Язык Python используется для обучения и в промышленности по всему миру. Он не только чрезвычайно мощный, но и очень простой в изучении. Более подробные сведения о Python — на сайте www.python.org[1].

[1] См. также: https://ru.wikipedia.org/wiki/Python и https://ru.wikibooks.org/wiki/Python/Учебник_Python_3.1. — *Примеч. пер.*

Приступая к изучению нового языка программирования или нового способа решения задач, компьютерные программисты всегда начинают с создания программы «Привет, Мир!». Это очень простая программа, которая выводит на экран фразу: «Привет, Мир!», чтобы убедиться, что все необходимое установлено, настроено и работает.

В Приключении 1 вы подготовите свой компьютер к созданию программы, выводящей текст «Hello Minecraft World» («Привет, мир Minecraft!») в чат Minecraft (рис. 1.1).

Рис. 1.1. «Привет, мир Minecraft!»

Чтобы писать программы для Minecraft, вам потребуется компьютер с установленной операционной системой Microsoft Windows. Для упрощения подготовки компьютера вы можете загрузить начальный набор инструментов с сайта https://adventuresinminecraft.github.io. Все инструменты проверены и гарантированно будут работать во всех приключениях, описанных в книге.

Обязательно загляните в файл README. В нем вы найдете список инструментов, вошедших в набор, и описание, как их установить. Эту информацию можно использовать для подготовки компьютера «с нуля», хотя мы не рекомендуем так поступать. Все, что нужно, вы сделаете, следуя инструкциям, данным в книге.

Очень важно правильно подготовить компьютер, иначе вы заплутаете в дебрях ошибок. Точно следуйте всем инструкциям, приведенным далее.

Подготовка компьютера

Прежде всего надо установить игру Minecraft на компьютер с Windows и запустить ее. Если у вас еще нет дистрибутива Minecraft и идентификатора пользователя, посетите веб-сайт www.minecraft.net и купите игру. Если столкнетесь с проблемами при установке, запуске или в ходе игры в Minecraft, обращайтесь за помощью по адресу: https://help.mojang.com[1].

Вам необходимо скачать Python и установить его на свой компьютер. В книге используется язык программирования Python версии 3. Программы были протестированы для работы с Python 3.6.1. Хотя вам и не обязательно (но рекомендуется) использовать именно эту версию, но у вас должен быть установлен Python не ниже версии 3.

Более подробные сведения о Python — на сайте www.python.org[2]. Загрузить Python можно с сайта www.python.org/download, а на Python Wiki (wiki.python.org) есть много информации о языке, учебники, а также ссылки на веб-сайты сообщества.

Чтобы подготовить персональный компьютер с операционной системой Windows к созданию первой программы для Minecraft, нужно выполнить три шага:

1. Загрузить набор инструментов для PC, в который входит предварительно настроенный сервер Minecraft с расширением RaspberryJuice и папка с именем *MyAdventures*, где вы будете сохранять свои программы для Minecraft.
2. Загрузить и установить язык программирования Python.
3. Настроить Minecraft и подключить игру к серверу.

На сайте книги www.wiley.com/go/adventuresinminecraft2e можно найти видеоурок, где демонстрируется порядок подготовки персонального компьютера с Windows.

Установка начального набора инструментов и Python

Итак, ниже описан первый этап подготовки компьютера, включающий в себя два шага:

1. Загрузка начального набора инструментов для Windows и извлечение его содержимого на рабочий стол.
2. Загрузка языка программирования Python и его установка.

---

[1] http://ru-minecraft.ru/skachat-minecraft/. — *Примеч. пер.*

[2] Смотрите также: https://ru.wikipedia.org/wiki/Python и https://ru.wikibooks.org/wiki/Python/Учебник_Python_3.1. — *Примеч. пер.*

Загрузка набора инструментов и извлечение его содержимого

Выполните следующие действия, чтобы загрузить начальный набор инструментов для Windows и скопировать его содержимое на рабочий стол. Размещение инструментов на рабочем столе упростит их поиск, когда они понадобятся.

1. Запустите веб-браузер (например, Internet Explorer или Chrome), перейдите на веб-сайт adventuresinminecraft.github.io и загрузите набор инструментов.
2. Когда загрузка zip-архива завершится, откройте его. Для этого откройте папку загрузки, щелкните правой кнопкой мыши на файле *AIMStarterKitPC.zip* и выберите в контекстном меню пункт Open File (Открыть) или просто дважды щелкните на этом файле.
3. Файл архива содержит единственную папку *AdventuresInMinecraft-PC*. Скопируйте ее на рабочий стол, для чего щелкните на папке левой кнопкой мыши и, удерживая кнопку, перетащите папку на рабочий стол (рис. 1.2.).

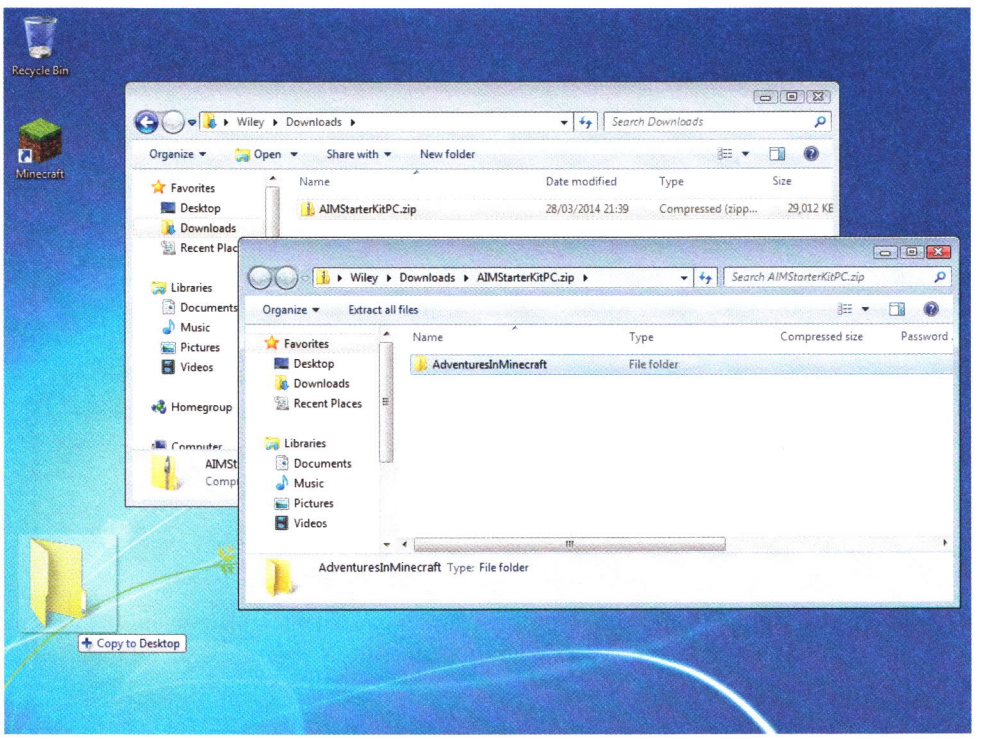


Рис. 1.2. Скопируйте папку AdventuresInMinecraft-PC на рабочий стол

Загрузка и установка Python

Поскольку вам предстоит писать программы на языке программирования Python, установите его и редактор IDLE, выполнив следующие шаги:

Если вы не имеете прав доступа администратора, придется попросить кого-то, обладающего такими правами, ввести пароль администратора, чтобы установить Python.

1. Запустите веб-браузер (например, Internet Explorer или Chrome). Перейдите на веб-сайт www.python.org/downloads/release/python-361/, чтобы скачать и установить Python 3.6.1.
2. Прокрутите страницу вниз и щелкните на ссылке Windows x86 Executable Installer в списке файлов для загрузки установщика Python.
3. Когда файл python-3.6.1.exe загрузится, запустите его, нажав в меню Open ⇨ Run, или откройте папку с загрузками и дважды щелкните на файле.
4. Может появиться окно с предупреждением «Вы хотите запустить этот файл?» («Do you want to run this file?»). Нажмите Run.
5. Щелкните на Install Now, чтобы начать установку.
6. Если система управления учетными записями пользователей (User Account Control) запрашивает разрешение на запуск установки программы, щелкните на Yes.
7. Дождитесь завершения установки программы и щелкните на Close.

Запуск Minecraft

В следующих приключениях мы будем просто предлагать запустить Minecraft. Если вы забудете, как запустить сервер и подключить к нему Minecraft, просто вернитесь к этому разделу.

Итак, вы установили на компьютер необходимое программное обеспечение. Но это не все: нужно запустить сервер и подключить к нему Minecraft — так, как описано ниже:

1. Откройте двойным щелчком мыши папку *AdventuresInMinecraft*, которую вы поместили на рабочий стол.
2. Запустите программу *StartServer*, дважды щелкнув на ней. Перед вами откроется командное окно.
3. Нажмите любую клавишу, чтобы запустить сервер. В процессе запуска будут выводиться разные строки, чтобы вы могли видеть, как идет работа.

Ваш компьютер может запросить разрешение на запуск сервера Minecraft или сообщить, что он не может быть открыт, так как это приложение от неизвестного разработчика или источника.

В Windows вам может потребоваться выбрать Дополнительную информацию (More) и Запустить в любом случае (Run anyway).

При первом запуске сервера на вашем компьютере может появиться сообщение с просьбой установить или обновить Java или разрешить серверу доступ к Сети. Если это произойдет, следуйте инструкциям по загрузке и установке Java или щелкните на кнопке Предоставить разрешение/доступ (Agree to give permission/access).

Если вы столкнулись с проблемами, обратитесь к справочному разделу на https://adventuresinminecraft.github.io.

4. Когда сервер завершит загрузку, в окне появится сообщение «Done» («Выполнено»), как показано на рис. 1.3.

```
"Adventures In Minecraft"
" Minecraft Server Version is 1.12"
"  Note - make sure Minecraft is using version 1.12"
"By continuing you are indicating your agreement to our EULA https://account.mojang.com/documents/minecraft_eula)."
Press any key to continue . . .
Loading libraries, please wait...
[20:49:31 INFO]: Starting minecraft server version 1.12
[20:49:31 INFO]: Loading properties
[20:49:31 INFO]: Default game type: CREATIVE
[20:49:31 INFO]: Generating keypair
[20:49:31 INFO]: Starting Minecraft server on *:25565
[20:49:31 INFO]: Using default channel type
[20:49:31 INFO]: This server is running CraftBukkit version git-Bukkit-ed8c725 (MC: 1.12) (Implementing API version 1.12
-R0.1-SNAPSHOT)
[20:49:32 INFO]: [RaspberryJuice] Loading RaspberryJuice v1.9
[20:49:32 WARN]: **** SERVER IS RUNNING IN OFFLINE/INSECURE MODE!
[20:49:32 WARN]: The server will make no attempt to authenticate usernames. Beware.
[20:49:32 WARN]: While this makes the game possible to play without internet access, it also opens up the ability for ha
ckers to connect with any username they choose.
[20:49:32 WARN]: To change this, set "online-mode" to "true" in the server.properties file.
[20:49:32 INFO]: Preparing level "world"
[20:49:32 INFO]: Preparing start region for level 0 (Seed: -176975456625273568)
[20:49:33 INFO]: Preparing spawn area: 30%
[20:49:34 INFO]: Preparing start region for level 1 (Seed: -176975456625273568)
[20:49:34 INFO]: Preparing start region for level 2 (Seed: -176975456625273568)
[20:49:35 INFO]: [RaspberryJuice] Enabling RaspberryJuice v1.9
[20:49:35 INFO]: [RaspberryJuice] Using port 4711
[20:49:35 INFO]: [RaspberryJuice] Using RELATIVE locations
[20:49:35 INFO]: [RaspberryJuice] ThreadListener Started
[20:49:35 INFO]: Server permissions file permissions.yml is empty, ignoring it
[20:49:35 INFO]: Done (3.038s)! For help, type "help" or "?"
>
```

Рис. 1.3. Командное окно сервера Minecraft с сообщением «Done» («Выполнено») после загрузки сервера

Версия Minecraft, используемая в начальном наборе, — версия 1.12. Это значит, что вам нужно создать профиль в загрузчике Minecraft для запуска версии 1.12 игры. Использование данной версии вместо последней гарантирует безупречную работу описанных программ. Сделать это нужно всего один раз (Minecraft запоминает, какую версию использовать).

Если вы хотите использовать более позднюю версию Minecraft, то изучите файл README в стартовом наборе, который описывает, как создать свой собственный стартовый набор; это следует делать только в том случае, если вы являетесь продвинутым пользователем, который хорошо разбирается в настройках компьютера, имеет опыт редактирования файлов конфигурации и опыт работы с программами Java. Проверяйте релизы новых стартовых наборов на странице *Adventures in Minecraft github* (https: // adventuresinminecraft.github.io).

Чтобы настроить загрузчик Minecraft на использование версии 1.12, выполните следующие шаги:

1. Запустите Minecraft и произведите вход, если требуется.
2. Когда появится окно *Minecraft Launcher* (Загрузчик Minecraft), щелкните на иконке в виде трех горизонтальных линий в правом врехнем углу экрана под именем пользователя и выберите Launch Options.
3. Щелкните на Add New, назовите вашу конфигурацию Adventures In Minecraft и выберите версию 1.12, как показано на рис. 1.4.

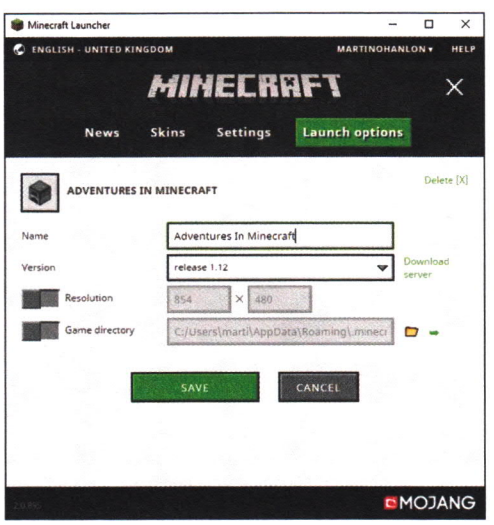


Рис. 1.4. Выбор используемой версии Minecraft

4. Щелкните на кнопке Save.
5. Щелкните на логотипе Minecraft и вернитесь в загрузчик.

Теперь подключите Minecraft к серверу:

1. Выберите профиль Adventures in Minecraft, щелкнув на стрелке вверх рядом с Play.
2. Щелкните на кнопке Play (Играть), чтобы запустить игру.
3. В меню игры выберите пункт Multiplayer (Сетевая игра).

При запуске игры может появиться запрос на подключение Minecraft к сети. Щелкните на кнопке Allow Access (Разрешить доступ).

4. В меню Play Multiplayer (Сетевая игра) выберите пункт Direct Connect (Прямое соединение).
5. Введите `localhost` в поле Server Address (Адрес сервера) и щелкните на кнопке Join Server (Подключиться к серверу), как показано на рис. 1.5. Вот и все! Теперь вы должны оказаться в мире Minecraft, созданном вашим сервером. С прибытием!

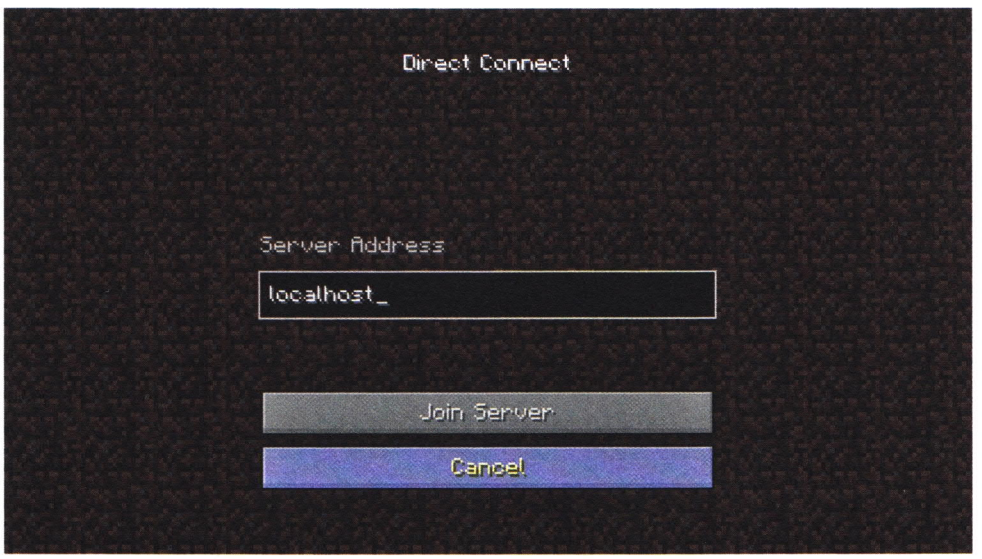


Рис. 1.5. Подключение Minecraft к серверу

Остановка сервера Minecraft

Завершив приключения в конце дня, нужно отсоединиться от сервера Minecraft. Для этого нажмите клавишу Escape в игре, чтобы выйти в меню, а затем щелкните на кнопке Disconnect (Отсоединиться).

Также следует безопасно отключить сервер, введя `stop` в командном окне сервера и нажав Enter.

По экрану побегут строчки, сообщающие, что происходит в процессе остановки.

В командном окне сервера можно вводить разные команды, управляющие разными параметрами игры Minecraft, такими как время суток или погода. Например, если в мире Minecraft начало темнеть, а вы еще не готовы к ночи, введите `time set 1` в командном окне, чтобы повернуть время назад, в утро. Если начнется дождь, очистить небо от туч позволит команда `weather clear`. Полный список команд можно получить по запросу `help`.

Создание программы

Поздравляем! Вы подготовили компьютер и запустили на нем игру Minecraft. Возможно, кому-то процесс подготовки показался слегка утомительным, но он уже позади. Повторить его придется только при желании использовать другой компьютер. Теперь пришло время приступить к самому интересному — созданию первой программы «Hello Minecraft World».

В будущих приключениях мы просто будем предлагать вам запустить IDLE. Если вы забудете, как это делается, просто вернитесь к данному разделу.

Прежде всего надо запустить Python и открыть IDLE.

Для этого щелкните на кнопке Start (Пуск) и выберите пункт меню Python 3.6 ▸ IDLE.

На экране должно появиться окно интерактивной оболочки Python.

При первом запуске IDLE может появиться запрос на запуск программы. В этом случае просто щелкните на кнопке, подтверждающей ваше согласие.

После того как откроется окно интерактивной оболочки Python, вы сможете создавать программы в IDLE. Программа, которую вам предстоит создать сейчас, едва ли поразит ваше воображение. Как программисты всегда начинают с программы «Привет, мир!», так и вы создадите программу «Hello Minecraft World», чтобы убедиться, что установка и настройка выполнены правильно. Для этого:

В следующих приключениях мы просто будем предлагать создать программу и сохранить ее в папке *MyAdventures*. Если вы забудете, как это делается, просто вернитесь к этому разделу.

1. Создайте новый файл, выбрав в меню IDLE пункт File ▸ New File (Файл ▸ Создать файл).
2. Сохраните файл в папке *MyAdventures*, выбрав в меню IDLE пункт File ▸ Save (Файл ▸ Сохранить).
3. Выберите папку *MyAdventure*:
   - в окне обозревателя, на панели навигации слева выберите пункт Desktop (Рабочий стол), дважды щелкните на папке *AdventuresInMinecraft-PC* и затем дважды щелкните на папке *MyAdventures*.
4. Далее присвойте своему новому файлу имя. Введите *HelloMinecraftWorld.py* и щелкните на кнопке Save (Сохранить). Расширение *.py* в конце сообщит компьютеру, что этот файл — программа на языке Python.

Важно сохранить программу в папке *MyAdventures*. Она специально предназначена для хранения программ из этой книги, поскольку содержит все необходимое для их работы.

5. Теперь пришло время начать программировать! Введите следующий программный код в окне редактора IDLE. Обратите внимание на правильный ввод прописных и строчных букв — Python различает буквы **разных регистров**:

```
import mcpi.minecraft as minecraft
mc = minecraft.Minecraft.create()
mc.postToChat("Hello Minecraft World")
```

6. Сохраните программу, выбрав в меню IDLE пункт File ▸ Save (Файл ▸ Сохранить).

Язык программирования Python **различает регистр символов**. Это значит, что вам следует правильно вводить прописные и строчные буквы. Например, Python будет поразному воспринимать слова `Minecraft` (с прописной буквы M) и `minecraft` (со строчной буквы). Если допустить ошибку при вводе, это приведет к ошибкам во время выполнения, и вам придется проверить уже пройденный путь, чтобы найти ошибку.

Подробнее о том, что означает этот код и что он делает, вы узнаете в следующем приключении. А пока вам достаточно знать, что он выводит на экран слова «Hello Minecraft World», чтобы показать, что всё работает правильно.

Запуск программы

В будущих приключениях мы просто будем предлагать запустить программу. Если вы забудете, как это делается, просто вернитесь к данному разделу.

Вы создали программу! Теперь самое время ее проверить. Для этого нужно сообщить оболочке IDLE, что она должна запустить программу. Для этого выполните следующие шаги:

1. Прежде чем запустить программу «Hello Minecraft World», нужно запустить и начать игру Minecraft. Если игра не запущена, сделайте это согласно описанному выше алгоритму.
2. Передвиньте окно игры Minecraft, чтобы вы могли одновременно видеть окна Minecraft и IDLE. (Если окно игры Minecraft открыто во весь экран, нажмите клавишу F11, чтобы выйти из полноэкранного режима.) На экране у вас должно быть четыре открытых окна (рис. 1.6):
   - окно интерактивной оболочки Python;
   - окно редактора IDLE с программой *HelloMinecraftWorld.py*;
   - командное окно сервера;
   - окно игры Minecraft.
3. Нажмите клавишу Esc, чтобы открыть игровое меню Minecraft. Щелкните мышью на окне редактора IDLE.

Рис. 1.6. Все готово к запуску программы

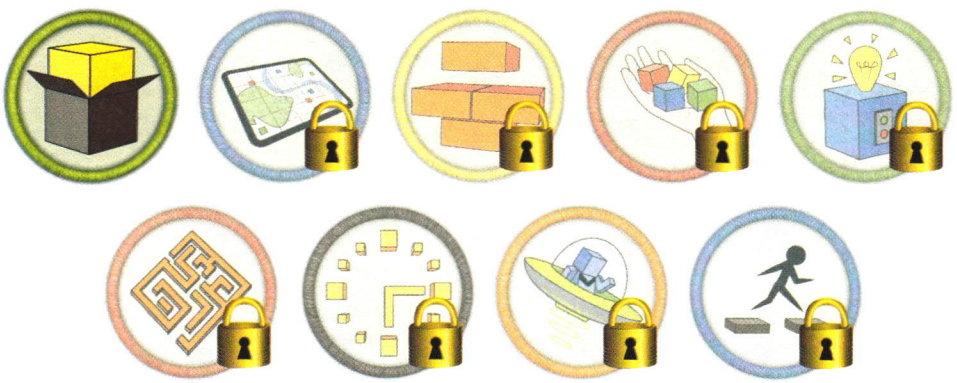
4. Запустите программу *HelloMinecraftWorld.py*, выбрав в меню IDLE пункт Run ▸ Run Module (Запустить ▸ Запустить модуль), или нажмите клавишу F5.
5. IDLE автоматически переключится в окно интерактивной оболочки Python и запустит программу. Если обнаружатся ошибки, сообщения о них будут выведены красным цветом. В этом случае внимательно проверьте введенный программный код, переключившись в окно редактора IDLE, и сравните его с программным кодом в книге.
6. Щелкните на заголовке окна с игрой Minecraft, чтобы вывести его на передний план. Ничего не заметили? В результате ваших немалых усилий в чате появилось сообщение «Hello Minecraft World». (Окно должно выглядеть примерно так, как показано на рис. 1.1.)

Если все подготовительные операции выполнены правильно, в чате Minecraft появится сообщение «Hello Minecraft World» (как на рис. 1.1). Если сообщение не появилось или в интерактивной оболочке Python появились сообщения об ошибках, вернитесь к началу этого приключения и повторите все инструкции. Важно, чтобы первый этап завершился успешно, иначе программы, созданные в следующих приключениях, просто не будут работать.

Остановка программы

В следующих приключениях мы будем просто предлагать остановить программу. Если вы забудете, как это делается, вернитесь к данному разделу.

Программа «Hello Minecraft World» запустилась, вывела сообщение на экран и затем сама остановилась. Но вам следует знать, как остановить программу принудительно, потому что в будущих приключениях вы будете создавать программы, которые остановятся только по вашему требованию! Чтобы остановить программу, щелкните на окне интерактивной оболочки Python и выберите пункт меню Shell ▸ Restart Shell (Оболочка ▸ Перезапустить оболочку) или, удерживая нажатой клавишу Ctrl, нажмите клавишу C.



Новое достижение: трудная часть завершена! Теперь у вас есть Minecraft, и вы написали свою первую программу. Мир Minecraft говорит вам «Привет!»

В СЛЕДУЮЩЕМ ПРИКЛЮЧЕНИИ

В Приключении 2 вы приобретете новые знания о программировании на языке Python для Minecraft, которые позволят вам определять позицию своего персонажа, познакомитесь с понятием «игра в игре» и напишете несколько простых игр, меняющих поведение в зависимости от местоположения персонажа в мире Minecraft.

Приключение 2
Наблюдение за перемещениями персонажа

ИГРАЯ в Minecraft, вы участвуете в игре, спроектированной и написанной другими людьми. Мир Minecraft удивителен, но он может стать еще более удивительным, если вы сумеете заставить его действовать так, как нужно вам. Написав первую программу для Minecraft на языке Python, вы получили в свои руки полноценную среду программирования, теперь можете придумать и запрограммировать все, что пожелаете. В этом Приключении вы познакомитесь с некоторыми интересными возможностями программирования на языке Python для Minecraft.

По-настоящему интересна возможность создавать игры внутри игры. Minecraft — это целый «мир», и ваши **программы** могут его менять. Далее вы узнаете, что большинство программ для Minecraft выполняют три основных действия: *извлекают* некоторую информацию о мире, такую как местоположение персонажа; *производят вычисления*, например считают заработанные очки; *вносят изменения*, например перемещают персонажа в другое место.

В этом Приключении вы увидите, как определить местоположение персонажа и воспроизвести разные события в игре в процессе его перемещения. В игре «Welcome home» («Добро пожаловать домой») вы создадите волшебный коврик, который будет приветствовать вас при касании его ногой. Затем мы познакомимся с приемом использования настоящих заборов в Minecraft, который называется *установка геозон* (geo-fencing), — на примере игры, которая предложит вам и вашим друзьям собрать максимальное количество объектов в кратчайшие сроки. И в заключение вы узнаете, как перемещать персонажа, написав игру «Катапульта», где ваш персонаж будет катапультироваться в небо, если не успеет покинуть поле в отведенный срок.

Программа — это последовательность инструкций или операторов на определенном языке программирования, которые автоматически выполняются компьютером. Инструкции должны записываться в соответствии с правилами используемого языка программирования. В этой книге используется язык Python.

Инструкция — универсальный компьютерный термин, обычно означающий одну законченную инструкцию, которую вы даете компьютеру, например, одна строка в программе, такая как `print("Hello Steve")`. Некоторые также используют слово *команда*, но оно больше подходит, когда речь идет о командах, вводимых в командной оболочке компьютера.

В языке Python есть собственное, очень точное определение инструкции, но в этой книге мы будем использовать слово «инструкция» для обозначения отдельных инструкций или строк в программах на языке Python. Желающие больше узнать о языке Python могут обратиться к электронной документации в интернете: https://docs.python.org/3/.

Определение позиции персонажа

Прежде чем вы сможете определять позицию персонажа, надо поближе познакомиться с организацией мира Minecraft. В нем все состоит из блоков; каждый блок является кубом с длиной ребра, равной одному метру. Когда Стив, персонаж в игре, перемещается по миру Minecraft, игра запоминает его позицию в виде набора **координат**. Мир состоит из пустых блоков, обычно в виде кубов с длиной ребра, равной одному метру, которые заполняются разными материалами. Для этого правила есть много исключений — например, покрытия и жидкости, такие как вода или лава, занимающие объем меньше блока и не обладающие способностью соединяться с другими материалами для заполнения куба целиком.

Определяя позицию персонажа в мире Minecraft, можно заставить игру реагировать на его перемещения. Например, выводить сообщения в случае попадания в определенное место, автоматически строить сооружения и менять позицию персонажа. Обо всем этом рассказано в книге, многое вы узнаете уже в этом Приключении!

Координата — набор чисел, однозначно определяющих местоположение. Для представления местоположений в трехмерном мире Minecraft используются трехмерные координаты, каждая из которых включает в себя три числа. Более подробную информацию о координатах можно найти по адресу: https://ru.wikipedia.org/wiki/Система_координат, а дополнительные сведения о системе координат в мире Minecraft — на http://ru.minecraft2.wikia.com/wiki/Координаты.

Координаты называются x, y и z. Будьте внимательны, говоря о координатах в Minecraft. Такие понятия, как «лево» и «право», в игре зависят от того, куда обращено лицо персонажа. Рассуждать о координатах в Minecraft лучше в терминах сторон света. На рис. 2.1 показано, как меняются x, y и z при перемещении в мире Minecraft.

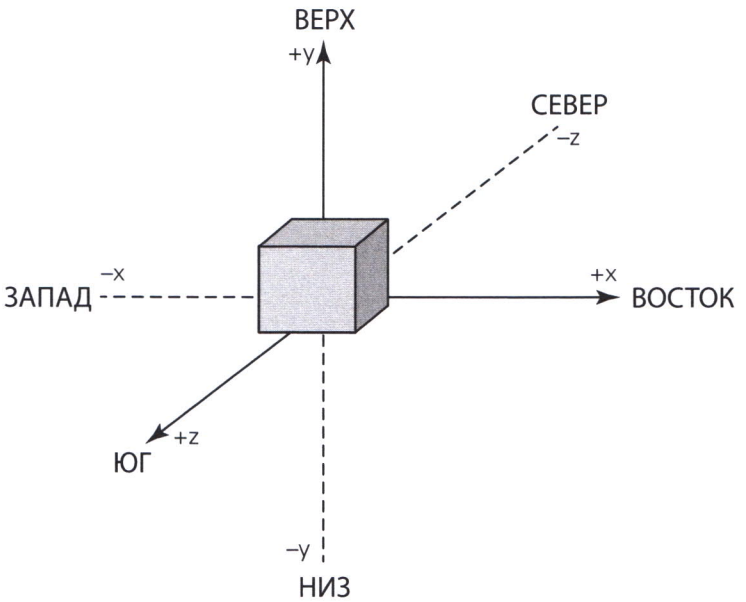
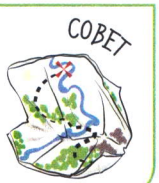


Рис. 2.1. Координаты в Minecraft тесно связаны со сторонами света



Чтобы было проще понять, как меняются координаты в мире Minecraft в процессе движения, поэкспериментируйте с ними во время игры. Для этого выведите их на экран нажатием клавиши F3. Описание других управляющих клавиш можно найти по адресу: http://minecraft.gamepedia.com/Controls.

Лучший способ понять координаты *x*, *y* и *z* — подумать о них с точки зрения компаса:

- значение *x* увеличивается при движении персонажа на восток и уменьшается при движении на запад;
- значение *y* увеличивается при движении персонажа вверх и уменьшается при движении вниз;
- значение *z* увеличивается при движении персонажа на юг и уменьшается при движении на север.

Начало

Теперь, когда у вас за плечами есть опыт создания первой программы «Hello Minecraft World» в Приключении 1, можно попробовать написать программу, устанавливающую связь с Minecraft!

В Приключении 1 подробно рассказывалось, какие шаги нужно сделать, чтобы все заработало. Вы будете повторять их в своих приключениях. Все этапы, которые необходимо преодолеть, чтобы приступить к вводу программы, описаны в Приключении 1. Тем не менее в первых нескольких Приключениях мы будем напоминать о них, пока вы учитесь, как это делать:

1. Запустите Minecraft, IDLE и сервер. Теперь вы знаете, как все запускается, но если вам потребуется освежить память, загляните в Приключение 1.
2. Откройте IDLE, среду разработки на Python, как вы уже делали в Приключении 1. Это окно, в которое вы будете вводить все программы на языке Python.

Чтобы вспомнить, как правильно подготовить и запустить Minecraft, посетите веб-сайт книги (www.wiley.com/go/adventuresinminecraft2e) и выберите видеоурок для Приключения 1.

Вывод позиции персонажа

Чтобы лучше понять, как устроена система координат в Minecraft, можно написать небольшую экспериментальную программу, чем мы и займемся прямо сейчас. Эта программа будет выводить на экран трехмерные координаты персонажа, благодаря чему вы сможете увидеть, как они меняются при движении в мире Minecraft.

1. В интерактивной оболочке Python создайте новую программу, выбрав пункт меню File ▸ New File (Файл ▸ Создать файл). На экране появится новое окно, куда вы будете вводить новую программу.
2. Привыкайте сохранять вновь созданный файл, прежде чем приступать к вводу программы, чтобы при срочной необходимости отойти от компьютера вам не пришлось придумывать имя для программы в спешке. Достаточно просто сохранить ее. Профессиональные программисты поступают именно так, чтобы не потерять работу из-за досадной случайности. В меню интерактивной оболочки Python выберите пункт File ▸ Save As (Файл ▸ Сохранить как) и введите имя файла *whereAmI.py*. Не забывайте, что программы нужно сохранять в папке *MyAdventures*.
3. Ваша программа будет взаимодействовать непосредственно с игрой Minecraft через Minecraft **API**. Чтобы получить доступ к этому API, требуется импортировать модуль `minecraft`, открывающий программам на языке Python доступ ко всем возможностям игры. Чтобы импортировать модуль, добавьте в программу следующую строку:

```
import mcpi.minecraft as minecraft
```

4. Чтобы обеспечить обмен данными с игрой Minecraft, нужно подключиться к ней. Для этого добавьте следующую строку в свою программу:

```
mc = minecraft.Minecraft.create()
```

Будьте внимательны: язык Python различает регистр символов, поэтому тщательно соблюдайте правила ввода прописных и строчных букв в именах. Это очень важно! Второе слово, `Minecraft`, в этой инструкции должно начинаться с прописной буквы, иначе она не будет работать.

5. Запросите у игры Minecraft позицию вашего персонажа с помощью инструкции `getTilePos()`:

```
pos = mc.player.getTilePos()
```

6. Запросите у Minecraft координаты персонажа. Инструкция `print()` выведет их в интерактивной оболочке Python:

```
print(pos.x)
print(pos.y)
print(pos.z)
```

7. Сохраните файл, выбрав в меню редактора пункт File ▸ Save (Файл ▸ Сохранить).
8. Запустите программу, выбрав в меню редактора пункт Run ▸ Run Module (Запустить ▸ Запустить модуль).

Теперь в окне интерактивной оболочки Python должны появиться координаты персонажа (рис. 2.2). Далее в этом Приключении вы будете использовать их для определения местоположения персонажа и конструирования волшебного коврика, приветствующего персонажа, когда тот на него наступит.

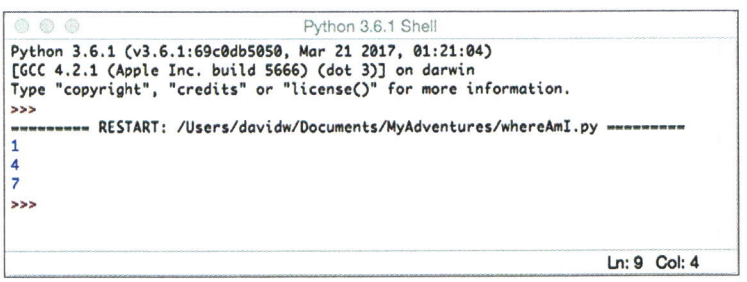


Рис. 2.2. Использование getTilePos для отображения координат персонажа в интерактивной оболочке Python

Аббревиатура **API** расшифровывается как *Application Programming Interface* (прикладной программный интерфейс). API открывает безопасный доступ к частям прикладной программы из ваших программ. В данном случае речь идет о Minecraft API (прикладном программном интерфейсе игры Minecraft), который открывает доступ к игре Minecraft из программ на языке Python. Игра Minecraft должна быть запущена до того, как программа на Python попытается использовать этот API. Имя `mcpi` в строке `import mcpi.minecraft as minecraft` означает *Minecraft Pi*, потому что первая версия API была написана для платформы Raspberry Pi.

Интерфейс — это набор правил, определяющих, как обращаться к разным частям компьютерной системы. API — это набор правил, определяющих порядок взаимодействия с выполняющейся программой, в данном случае — с игрой Minecraft. Все ваши программы для Minecraft будут обращаться к запущенной игре посредством Minecraft API.

Всем программам на языке Python, которые вы будете создавать, потребуется доступ к Minecraft API. Для безошибочной работы вашим программам необходим модуль `mcpi.minecraft`, который хранится в папке *mcpi*, внутри папки *MyAdventures*. Самый простой способ гарантировать доступность этого модуля — всегда создавать свои программы на Python в папке *MyAdventures*.

УГЛУБЛЯЕМСЯ В КОД

Поздравляем! Вы только что использовали переменные в своей программе: `pos.x` — это переменная, `pos.y` и `pos.z` — тоже.

Переменная — имя области в памяти компьютера. Всякий раз, когда имя переменной помещается слева от знака равенства, в ней сохраняется указанное значение, например:

```
a = 10
```

Всякий раз, когда имя переменной используется внутри инструкции `print`, программа выводит ее значение в интерактивной оболочке Python. То есть если `a = 10`, следующая инструкция:

```
print(a)
```

выведет число `10`, значение переменной `a`.

Переменная `pos`, которая используется в инструкции `pos = mc.player.getTilePos()`, это переменная особого типа. Представьте себе ящик с тремя отсеками, помеченными `x`, `y` и `z`:

```
print(pos.x)
```

Попробуйте переместить персонаж в другое место и вновь запустить программу. Посмотрите, какие координаты на этот раз выводятся в интерактивной оболочке Python: они должны измениться в соответствии с новым местоположением. Вы можете использовать эту маленькую программу всякий раз, когда вам требуется узнать координаты персонажа в мире Minecraft.

Переменная — это имя области в памяти компьютера. Сохранять новые значения в переменных можно в любом месте программы, а также читать их значения, выводить на экран или использовать в вычислениях. Переменные можно считать своеобразной кнопкой «Память» на калькуляторе.

Улучшение отображения координат

Прежде чем двинуться дальше, давайте улучшим вывод позиции персонажа, чтобы сделать его чуть понятнее. Вывод в трех строках занимает слишком много места; вы могли бы вывести те же координаты в одной строке, как показано ниже:

```
x=10 y=2 z=20
```

Для этого выполните следующие шаги:

1. Удалите из программы *whereAmI.py* три инструкции `print()` и замените их следующей строкой:

   ```
   print("x="+str(pos.x) +" y="+str(pos.y) +" z="+str(pos.z))
   ```

2. Сохраните программу, выбрав в меню редактора пункт File ▸ Save (Файл ▸ Сохранить), и запустите ее, выбрав в меню пункт Run ▸ Run Module (Запустить ▸ Запустить модуль). Посмотрите, насколько понятнее стал вывод — теперь все три координаты выводятся в одной строке!

УГЛУБЛЯЕМСЯ В КОД

В новой версии программы *whereAmI.py* появилось немного волшебства, которое нам хотелось бы объяснить.

Инструкция `print()` в языке Python выводит все, что найдет между круглыми скобками, в интерактивную оболочку Python. Вы уже использовали разные способы отображения информации с помощью `print()`. Теперь мы займемся их исследованием. Взгляните на следующий пример:

```
print("hello")
```

Как в программе «Hello Minecraft World», здесь инструкция `print()` выведет слово *hello* (привет). Кавычки сообщают интерпретатору Python, что требуется вывести слово в точности так, как оно напечатано. Если добавить пробелы внутри кавычек, они также будут выведены. Текст между кавычками называется **строкой**.

Вот другой пример использования `print()`:

```
print(pos.x)
```

В этом случае `print()` выведет значение, которое хранится в переменной `pos.x`. Это число, но оно может меняться всякий раз, когда используется `print()` — в зависимости от значения, хранившегося в переменной в этот момент.

А теперь разберемся, что делает `str()` внутри инструкции `print()`. Инструкция `print()` в языке Python позволяет смешивать числа и строки в любых комбинациях. Когда в программе требуется смешать числа и строки, как в следующем примере, нужно сообщить интерпретатору Python, что тот должен преобразовать числа в строки, чтобы их можно было объединить с другими отображаемыми строками. `str()` — встроенная инструкция языка Python, она преобразует любую переменную или значение, помещенные в круглые скобки, в строку. Символ «плюс» (`+`) сообщает интерпретатору, что он должен объединить строку с `"x= "` с содержимым переменной `pos.x` в одну большую строку, которая затем будет выведена в окно интерактивной оболочки Python:

```
print("x= "+str(pos.x))
```

Почему бы вам не попробовать выполнить то же самое, но без `str()`? Посмотрите, что получится:

```
print("x= "+ pos.x)
```

Строка — тип переменных, способных хранить последовательности букв, цифр и других символов. Он получил такое название, потому что напоминает строку в книге. Как и строки в книге, строковые переменные могут включать в себя буквы, цифры, знаки препинания и другие символы. Символы в таких переменных хранятся в том же порядке, в каком вы их напечатали.

Строки можно использовать для разных целей — например, для хранения имен, адресов или сообщений для вывода в чат Minecraft.

Использование postToChat для вывода координат в чат

Не очень удобно, что программа *whereAmI.py* выводит координаты в окно интерактивной оболочки Python, тогда как игровые события происходят в окне Minecraft. Однако этот недостаток легко исправить, воспользовавшись приемом, уже известным вам по Приключению 1: `postToChat`. Для этого

1. Замените инструкцию `print()` на `mc.postToChat()`, как показано ниже:

```
mc.postToChat("x="+str(pos.x) +" y="+str(pos.y) ↩
  +" z="+str(pos.z))
```

2. Выберите пункт меню File ▸ Save (Файл ▸ Сохранить), чтобы сохранить программу, и запустите ее, выбрав пункт меню Run ▸ Run Module (Запустить ▸ Запустить модуль).

Отлично! Теперь координаты персонажа должны появиться в чате Minecraft, что намного удобнее для игрока.

Введение в игровой цикл

Необходимость запускать программу *whereAmI.py* всякий раз, когда понадобится узнать координаты персонажа, делает эту программу менее полезной. К счастью, проблему легко решить, добавив в программу игровой цикл. Практически все программы, которые вам предстоит написать при чтении этой книги, тоже используют игровой цикл, чтобы обеспечить непрерывную работу программы и ее взаимодействие с игрой Minecraft. Большинство игр продолжают выполняться, пока вы не закроете их, это очень полезное свойство. В программировании такое поведение называется **бесконечным циклом** (такие циклы никогда не завершаются).

Бесконечный цикл — цикл, который продолжает выполняться снова и снова, до бесконечности. Единственный способ прервать бесконечный цикл — остановить программу на Python. Сделать это можно из интерактивной оболочки IDLE, выбрав пункт меню Shell ▸ Restart Shell (Оболочка ▸ Перезапустить оболочку) или нажав комбинацию клавиш CTRL + C на клавиатуре.

Вы можете добавить игровой цикл, изменив уже имеющуюся программу. Новая программа очень похожа на предыдущую, поэтому такое решение сэкономит вам время. Не забудьте сохранить новую программу под другим именем, чтобы не потерять первую.

1. Сначала сохраните программу в новом файле с именем *whereAmI2.py*, выбрав в меню редактора пункт File ▸ Save As (Файл ▸ Сохранить как). Обязательно сохраните ее в папке *MyAdventures*, иначе она не будет работать!

   Теперь нужно импортировать новый модуль, с помощью которого можно на время приостанавливать работу программы. Это необходимо, иначе если не замедлить цикл, он засыплет чат сообщениями, и вы не сможете видеть, что происходит в игре. Вы можете быстро вставлять отступы, нажимая клавишу Tab на клавиатуре. Добавьте в свою программу строку, выделенную **жирным шрифтом**:

```
import mcpi.minecraft as minecraft
import time
```

2. Измените основную часть программы, добавив строки, выделенные **жирным** в следующем листинге, и обратите особое внимание на отступы в строках ниже инструкции `while True:` — они отличают строки, которые должны повторяться в цикле. Подробнее, зачем нужны эти отступы, рассказано ниже, во врезке «Углубляемся в код»:

```
while True:
  time.sleep(1)
  pos = mc.player.getTilePos()
  mc.postToChat("x="+str(pos.x) +" y="+str(pos.y) +" ↩
   z="+str(pos.z))
```

3. Выберите в меню редактора пункт File ▸ Save (Файл ▸ Сохранить), чтобы сохранить изменения.
4. Теперь можно запустить программу, выбрав в меню редактора пункт Run ▸ Run Module (Запустить ▸ Запустить модуль). Побродите по миру Minecraft и обратите внимание на то, что каждую секунду координаты персонажа отправляются в чат.

УГЛУБЛЯЕМСЯ В КОД

Отступы играют важную роль при программировании на любом языке, так как позволяют обозначить структуру программы и выделить группы инструкций, принадлежащих другим инструкциям. Но в языке Python отступы играют еще более важную роль: они помогают интерпретатору Python понять смысл вашей программы (в отличие от других языков, таких как C, где для группировки инструкций используются фигурные скобки `{}`). Отступы важны в циклах `while`, так как показывают, какие инструкции принадлежат циклу и должны повторяться. В данном игровом цикле программные инструкции, находящиеся ниже `while True:`, имеют отступы, потому что все они принадлежат циклу и должны выполняться в каждой его итерации.

В примере ниже можно видеть, что слово «hello» (привет) выводится один раз, а слово «tick» (щелк) — каждую секунду. Инструкции `time.sleep()` и `print()` принадлежат циклу, потому что они имеют отступ:

```
print("hello")
while True: # это начало цикла
  time.sleep(1)
  print("tick")
```

Отступ — это пространство слева в каждой строке программы. Отступы используются, чтобы обозначить структуру программы и сгруппировать программные инструкции в циклах и других инструкциях. В языке Python отступы играют еще более важную роль, так как определяют смысл программы.

Очень важно, чтобы вы правильно оформляли отступы. Язык Python понимает отступы, оформленные пробелами и символами табуляции, но их нельзя смешивать. Если символы табуляции появятся вместе с пробелами в одной программе, Python может не понять, что вы имели в виду. Поэтому всегда используйте что-то одно для оформления отступов. Большинство предпочитает использовать клавишу табуляции Tab — это проще, чем вводить множество пробелов. Но если вы начали использовать клавишу Tab, делайте это постоянно и не смешивайте табуляцию с пробелами.

Теперь, когда ваша программа выполняет бесконечный игровой цикл, она никогда не остановится самостоятельно! Прежде чем запустить другую программу, сначала надо остановить эту, выбрав в интерактивной оболочке Python пункт меню Shell ▸ Restart Shell (Оболочка ▸ Перезапустить оболочку) или нажав комбинацию клавиш CTRL+C на клавиатуре.

Создание игры Welcome Home

Теперь пришло время взять на вооружение игровой цикл и создать на его основе простое приложение. Игра «Welcome Home» будет следить за позицией персонажа по ходу его перемещения в мире Minecraft. В процессе разработки этой программы вы также узнаете, как принимать сложные решения с помощью инструкции `if`, и создадите волшебный коврик, который будет выводить сообщение «welcome home» («Добро пожаловать домой») в чат Minecraft, когда персонаж встанет на него.

Чтобы увидеть, как написать игру «Welcome Home» и играть в нее, посетите веб-сайт книги (www.wiley.com/go/adventuresinminecraft2e) и выберите видеоурок для Приключения 2.

Использование инструкций if для создания волшебного коврика

Чтобы поймать момент, когда персонаж встанет на коврик, используйте инструкцию `if`: она позволит сравнить координаты персонажа с координатами коврика. При их совпадении можно смело утверждать, что персонаж пришел домой.

Для начала посмотрим, как действует инструкция `if`. Для этого выполните следующие шаги в интерактивной оболочке Python:

1. Щелкните на окне интерактивной оболочки Python. Щелчок следует выполнить чуть правее приглашения к вводу `>>>`, чтобы сразу начать ввод в нужном месте.
2. Введите следующую строку и нажмите клавишу Enter, чтобы Python выполнил ее. В результате ничего не будет выведено, так как эта инструкция просто записывает число 20 в переменную с именем `a`:

   ```
   a = 20
   ```

3. Проверьте, записалось ли число в переменную `a`, для чего введите строку:

   ```
   print(a)
   ```

   Она должна вывести число 20.

4. С помощью инструкции `if` определите, действительно ли значение переменной `a` больше 10. Обратите внимание на двоеточие в конце инструкции `if`. Когда вы нажмете клавишу Enter в конце строки с инструкцией `if`, Python автоматически добавит отступ в начале следующей строки:

```
if a>10:
  print("big")
```

5. Теперь нажмите клавишу Enter еще раз, чтобы сообщить интерактивной оболочке Python, что ввод инструкции `if` закончен. В результате должно появиться слово «big» (большое). На рис. 2.3 показано, как выглядит происходящее.

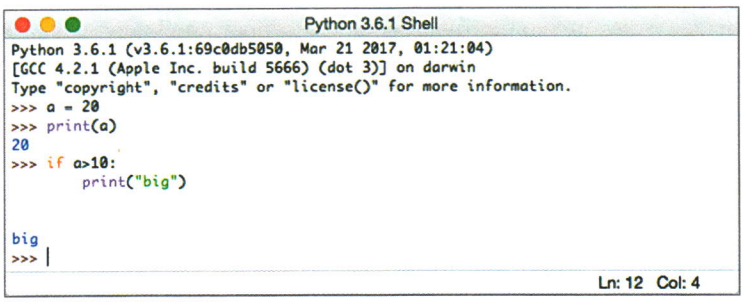


Рис. 2.3. Использование инструкции if в интерактивной оболочке Python

Инструкция `if` дает два результата. В примере, приведенном выше, инструкция `if a>10` может иметь истинный (`True`, если `a` больше 10) или ложный (`False`, если `a` не больше 10) результат. Только один из двух! Со значениями `True` и `False` вы еще встретитесь далее в этом Приключении.

Проверка нахождения персонажа в определенном месте

Чтобы определить момент, когда персонаж встанет на коврик, программа должна проверить минимум две координаты. Достаточно проверить совпадение координат x и z персонажа и коврика, чтобы сказать, стоит ли первый на втором. Проверить совпадение одной u другой координаты можно с помощью ключевого слова `and` (переводится как «и») внутри инструкции `if`. Координата y в данном случае менее важна, поэтому программа не будет ее проверять: даже если персонаж будет парить над ковриком, на экране появится приветственное сообщение.

Попробуйте выполнить следующие шаги в интерактивной оболочке Python, чтобы проверить, как работает этот метод:

1. В строке приглашения к вводу `>>>` введите следующую переменную:

```
a = 8
```

2. Введите инструкцию `if`, в которой также используется ключевое слово `and`. Эта инструкция проверит, находится ли переменная `a` в диапазоне между двумя числами. Сразу после ввода инструкции вы увидите, как появится слово «between» (между): число 8 больше 5 и меньше 10. Не забудьте нажать Enter повторно, чтобы завершить строку с отступом:

```
if a>5 and a<10:
  print("between")
```

3. Теперь измените значение `a`, чтобы оно было меньше 5, и посмотрите, что произойдет (рис. 2.4):

```
a = 3
if a>5 and a<10:
  print("between")
```

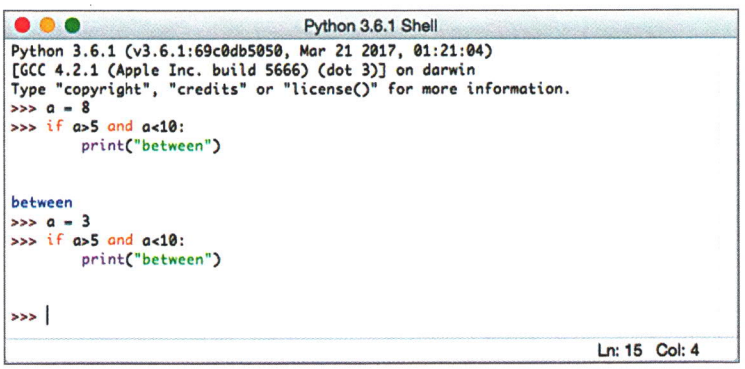


Рис. 2.4. Использование инструкции if с ключевым словом and для проверки двух и более условий

Попробуйте предыдущий пример с разными числами, особенно с близкими к 5 и 10. Для каких чисел будет выводиться слово «between»?

Когда вы тестируете программу, полезно сделать это с максимально возможным количеством разных чисел. Чтобы уменьшить число проверок, которых может быть бесконечно много, внимательно посмотрите на условия в инструкциях `if` и выполните проверки только для чисел, близких к указанным в условиях. В предыдущем примере вполне можно ограничиться проверкой чисел 4, 5, 6 и 9, 10, 11. Если для них получаются ожидаемые результаты, легко предположить, что для любых других чисел результаты тоже получатся правильными.

Создание волшебного коврика

Прежде чем написать программу, в мире Minecraft нужно создать нечто, с чем программа будет взаимодействовать. Для текущего Приключения нам нужен коврик. Поэтому запустите игру Minecraft и загрузите мир, который вы уже использовали.

1. Чтобы положить коврик перед дверью, выберите что-нибудь из имеющихся предметов и щелкните правой кнопкой на месте перед персонажем. Шерсть прекрасно подойдет на роль коврика.
2. Чтобы определить координаты коврика в мире Minecraft, запустите программу *whereAmI.py* и затем поставьте персонажа на коврик. Запишите куда-нибудь координаты *x*, *y* и *z* (они вам потребуются, когда будете писать новую программу).

Возможно, вам не терпится приступить к созданию больших сооружений. В Приключении 3 вы узнаете, как это делать автоматически, с помощью программ на языке Python, а пока мы будем строить простые сооружения и сосредоточимся на создании программ, работающих без ошибок. Если вам захочется построить простой дом перед ковриком — стройте, но не забудьте, что обычно коврик лежит перед дверью, при входе в дом.

Игра Welcome Home

Теперь, когда вы знаете, как работает инструкция `if`, можно приступать к созданию игры «Welcome Home» («Добро пожаловать домой»). Для этого выполните следующие шаги:

1. Выберите в меню интерактивной оболочки Python пункт File ▸ New File (Файл ▸ Создать файл), чтобы создать новую программу.
2. Выберите в меню редактора пункт File ▸ Save As (Файл ▸ Сохранить как), чтобы сохранить программу, и назовите ее *welcomeHome.py*. Не забывайте, что программы нужно сохранять в папку *MyAdventures*, иначе они не будут работать.
3. Импортируйте модули, необходимые программе:

```
import mcpi.minecraft as minecraft
import time
```

4. Подключите программу к игре Minecraft, помня о корректном вводе заглавных и прописных букв в слове `Minecraft`:

```
mc = minecraft.Minecraft.create()
```

5. Добавьте игровой цикл, определяющий позицию персонажа, с задержкой, чтобы он выполнялся не слишком быстро:

```
while True:
  time.sleep(1)
  pos = mc.player.getTilePos()
```

6. Добавьте инструкцию `if`, проверяющую, не встал ли персонаж на коврик. Здесь в инструкции `if` используется ключевое слово `and` для проверки двух условий. Чтобы программа вывела приветственное сообщение, обе координаты коврика — *x* и *z* — должны

совпасть с координатами персонажа. У вас сохранились координаты *x* и *z* коврика, которые вы должны были записать? Введите их здесь, чтобы программа знала, где у вас находится коврик:

```
if pos.x == 10 and pos.z == 12:
  mc.postToChat("welcome home")
```

Теперь посмотрим, работает ли программа! Выберите в меню редактора пункт Run ▸ Run Module (Запустить ▸ Запустить модуль) и походите по миру Minecraft. Когда персонаж окажется на коврике, программа должна вывести «welcome home» (добро пожаловать домой), как показано на рис. 2.5. Круто!

Рис. 2.5. Если встать на коврик, появится сообщение «welcome home»

Надеюсь, вы заметили, что в инструкции `if` используется символ `==` (два знака равенства). Начинающие программисты часто ошибаются, используя один знак `=` вместо двух. Попробуйте допустить такую же ошибку и посмотрите, что получится: при попытке запустить программу вы увидите сообщение об ошибке. В языке Python один знак равенства (`=`) означает: «записать значение справа в переменную слева», например, `a = 3`. Два знака равенства (`==`) в языке Python означают: «проверить равенство значений слева и справа», например, `if a == 3:`.

Не забудьте правильно оформить отступы в программе «Welcome Home». Инструкции, являющиеся частью цикла `while True,` должны иметь один отступ. Инструкция `mc.postToChat()`, принадлежащая инструкции `if`, должна иметь два отступа. Если отступы оформлены неправильно, программа тоже будет работать неправильно.

Хотите верьте, хотите нет, но вы узнали самое необходимое: все то, что понадобится в следующих приключениях. Все программы для Minecraft импортируют модули, подключаются к игре, содержат игровой цикл, получают какую-то информацию из игры и действуют в соответствии с ней. На основе этих простых правил вы сможете писать умопомрачительные программы!

Как вы думаете, проверка координаты *y* (высота местоположения) поможет что-то улучшить в определении момента, когда персонаж встает на коврик? Попробуйте! Измените программу так, чтобы она проверяла совпадение всех трех координат персонажа с координатами коврика, и посмотрите, будет ли приложение работать лучше.

УГЛУБЛЯЕМСЯ В КОД

Возможно, в какой-то момент вы допустите ошибку при вводе программы. Тогда вы увидите в интерактивной оболочке Python красное сообщение об ошибке. Для начала взгляните на ее тип и не пугайтесь, видя такие сообщения время от времени.

Если ввести неправильные символы или ввести их в неправильном порядке, интерпретатор сообщит о **синтаксической** ошибке (`SyntaxError`). Если пропустить какой-то символ или ввести имя переменной там, где ему не место, интерпретатор тоже просигналит о синтаксической ошибке.

Например, если ввести следующую строку в интерактивной оболочке Python, можно увидеть синтаксическую ошибку, как показано на рис. 2.6, потому что знак равенства сам по себе не является самостоятельной инструкцией, обычно он используется для записи значений в переменные:

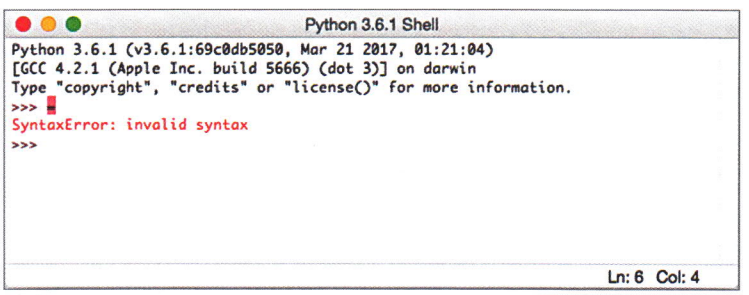


Рис. 2.6. Python показывает, где в коде обнаружилась ошибка

Теперь введите в оболочке Python:

```
print(zz)
```

Вы увидите, что Python обнаружил ошибку в имени (рис. 2.7): в переменной с именем `zz` не сохранялось никаких значений, соответственно, Python ничего о ней не знает.

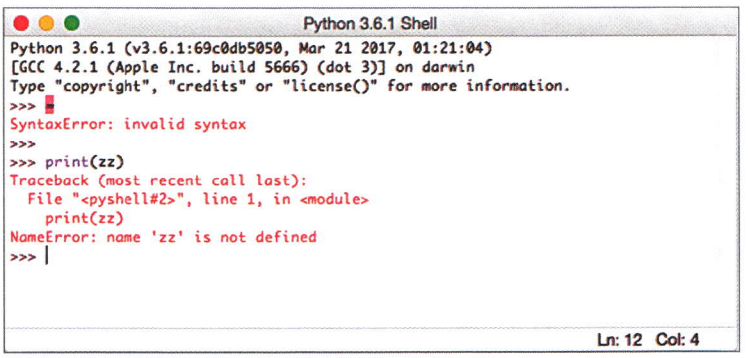


Рис. 2.7. Python сообщает об ошибке в имени

Попробуйте ввести цикл `while` без важного двоеточия (`:`) в конце:

```
while True
```

Python опять сообщит о синтаксической ошибке, потому что ожидает увидеть двоеточие, которое вы пропустили.

Не пугайтесь, видя сообщения об ошибках, просто вернитесь к своей программе. Тщательно рассмотрите строку, в которой Python нашел ошибку, и попытайтесь определить, где вы допустили опечатку. Иногда вы будете получать сообщения о синтаксических ошибках, которые в действительности находятся не в той строке, на которую указывает Python, а выше. Поэтому обязательно проверьте соседние строки: проблема может скрываться в них. Если вам не удалось найти причину ошибки, попросите друга взглянуть на программу. Часто ошибки легче найти в чужой программе, чем в собственной! Когда интерактивная оболочка Python сообщает об ошибке, она добавляет в текст сообщения номер строки. Посмотрите справа внизу, в окне IDLE, и увидите номер строки, в которой находится курсор, или выберите в меню IDLE пункт Edit ▸ GoTo Line (Правка ▸ Перейти к строке).

Синтаксис — свод правил языка (в данном случае языка Python), главным образом связанных с порядком ввода символов.

Использование геозон

Теперь вы будете писать новую игру с названием *rent.py*. В этой игре есть поле, окруженное забором. Ваша программа определит, находится ли персонаж на поле, и будет снимать очки, пока он его не покинет. Задача игрока — максимально быстро собрать все предметы, имеющиеся на поле, чтобы потерять как можно меньше очков.

В предыдущей программе *welcomeHome.py* вы использовали инструкцию `if`, чтобы определить, совпадают ли координаты персонажа с координатами коврика. Чтобы совпадение обнаружилось, персонаж должен был встать точно на коврик. Вы можете улучшить проверку, воспользовавшись приемом, который называется **установка геозон** (geo-fencing) и используется в новой игре. В этой программе надо расширить проверку местоположения персонажа, потому что на сей раз требуется проверить его присутствие в некоей области мира Minecraft, а не на конкретной позиции. Данный прием позволит обнаружить местоположение персонажа в любой точке обширной области, а не только на конкретном блоке.

Установка геозон (**geo-fencing**) — универсальный прием построения виртуального забора вокруг координат на любых картах. Если кто-то заходит в огражденную область, происходят какие-то события. Этим «кто-то» может быть персонаж в мире Minecraft, человек — в реальности, устройство (например, газонокосилка) или животное.

Часто геозоны используются, чтобы предупредить пользователя: кто-то или что-то — например, корова или машина, — оказывается за пределами определенного участка. Больше информации об использовании геозон в реальном мире можно найти в Википедии: https://ru.wikipedia.org/wiki/Геозоны, а также по адресу: http://www.cbsnews.com/news/kenya-usestext-messages-to-track-elephant/, где рассказано об использовании геозон для слежения за перемещениями диких слонов.

Чтобы установить геозону, необходимы: объект или область для отслеживания, допустимые координаты «забора» вокруг объекта. Далее вы узнаете, как это реализовать, построив поле с настоящим забором вокруг него в мире Minecraft и описав координаты углов поля.

Для работы этой программы забор сам по себе не нужен, но он сделает игру интереснее, потому что вам придется перепрыгнуть через забор или обойти его вокруг, чтобы добраться до прохода и покинуть поле. Добавление препятствий в игры делает их сложнее и интереснее. Забор Minecraft будет построен вдоль границ геозоны, по заданным вами координатам, а наличие геозоны даст вашей программе простую возможность ответить на вопрос: «Персонаж на поле или нет?»

Когда вы построите поле и забор, они будут выглядеть так, как показано на рис. 2.8.

Обработка координат углов поля

Чтобы определить в программе геозону для поля, нужно знать точные координаты его углов и ввести их в программу на языке Python.

Рис. 2.8. Поле, огороженное забором. Поле имеет размер 10 × 10 блоков

Самый простой способ сделать это — запустить программу *whereAmI.py*, написанную ранее, переместить персонажа последовательно в каждый угол будущего поля и записать координаты *x*, *y* и *z* каждого угла. Нарисуйте на листе бумаги план поля и запишите на нем координаты углов. Они вам понадобятся, когда будете писать программу. Не забывайте, что координата *z* увеличивается, когда персонаж движется на юг, и уменьшается при движении на север, как показано на рис. 2.1.

На рис. 2.9 даны координаты углов поля, которое мы построили, когда писали книгу. Мы записали координаты всех четырех углов, так как они нужны для определения наибольших и наименьших значений в обоих направлениях — *x* и *z*. Обратите внимание, что наименьшие значения координат находятся в верхнем левом углу поля. Однако у вас наименьшие значения могут быть в другой части поля, если они были заданы с персонажем, обращенным в другую сторону.

Для создания геозоны вам потребуется четыре числа. Во-первых, надо узнать наибольшую и наименьшую координаты *x*, записанные на вашем плане. В предыдущем примере наименьшая координата *x* имела значение 10, наибольшая — 20. Наименьшую координату *x* я назвал `X1`, наибольшую — `X2`.

В зависимости от того, где находится персонаж в мире Minecraft, некоторые координаты могут оказаться отрицательными (например, `x=-5`, `y=0`, `z=-2`), что может осложнить вычисления. Будет проще, если вы перейдете в ту часть в мире Minecraft, где все три координаты положительные (например, `x=5`, `y=2`, `z=4`). Увидеть координаты персонажа можно в левом верхнем углу окна игры Minecraft, если нажать клавишу F3.

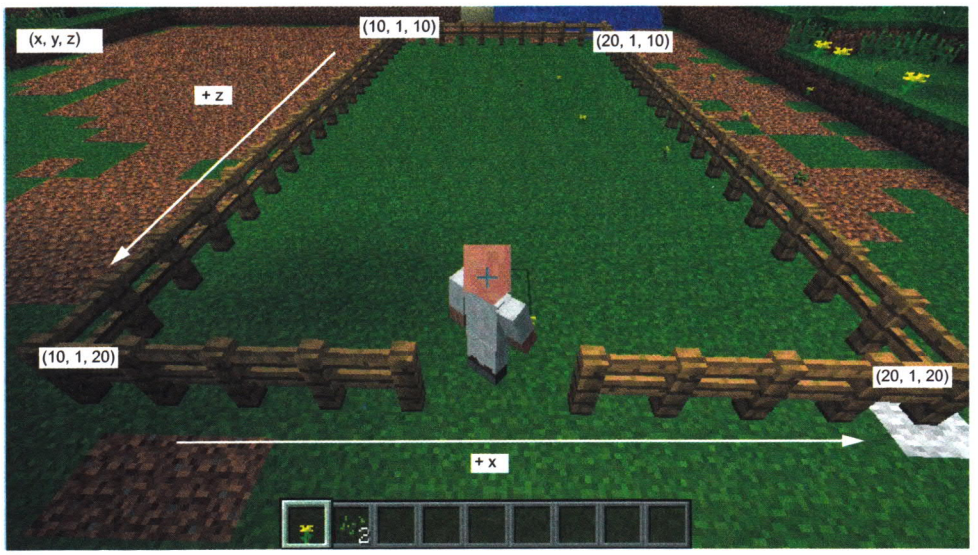


Рис. 2.9. После получения координат углов поля можно приступать к определению геозоны

Далее запишите наименьшую и наибольшую координаты *z*. В данном примере наименьшая координата *z* имеет значение 10, наибольшая — 20. Наименьшую координату *z* я назвал `Z1`, наибольшую — `Z2`:

```
X1 = 10
Z1 = 10
X2 = 20
Z2 = 20
```

Запишите наибольшие и наименьшие координаты. Они вам понадобятся, когда будете писать программу.

Создание программы установки геозоны

Заключительный этап — создание программы, использующей геозону и отнимающей очки, когда персонаж оказывается на поле. По своей структуре она напоминает программу *whereAmI.py*.

В этой программе также будут использованы **константы**, благодаря которым проще перемещать поле в другое место. Вам не придется просматривать текст программы, чтобы найти все значения координат, требующие изменения для переноса поля.

Константа — это имя области в памяти компьютера (как и переменная), где можно хранить значения, которые обычно не меняются в ходе выполнения программы.

В отличие от других языков программирования, в Python константы ничем не отличаются от переменных. Программисты на Python обычно используют соглашение, согласно

которому имена констант состоят только из заглавных букв. Такие имена наглядно показывают, что являются константами и не должны изменяться после присвоения начального значения. В других языках программирования константы обрабатываются иначе; в них попытка изменить константу приводит к ошибке. В Python такая особенность отсутствует.

Выполните следующие шаги, чтобы написать программу:

1. Откройте новое окно IDLE, выбрав в меню пункт File ▸ New File (Файл ▸ Создать файл).
2. Выберите пункт меню File ▸ Save As (Файл ▸ Сохранить как) и сохраните файл программы с именем *rent.py*.
3. Импортируйте необходимые модули:

```
import mcpi.minecraft as minecraft
import time
```

4. Подключитесь к игре Minecraft:

```
mc = minecraft.Minecraft.create()
```

5. Определите четыре константы с координатами для создания геозоны. Убедитесь, что используете здесь координаты, определенные ранее. Я использовал координаты, полученные в моем случае; ваши могут отличаться:

```
X1 = 10
Z1 = 10
X2 = 20
Z2 = 20
```

6. Создайте переменную, в которой будет накапливаться количество очков, потерянных персонажем во время нахождения на поле. В начале игры персонаж еще не потерял ни одного очка, поэтому создайте новую переменную `rent`, присвоив ей нулевое значение:

```
rent = 0
```

7. Создайте главный игровой цикл с помощью инструкции `while`. Цикл будет совершать по одной итерации в секунду. Поэтому добавьте инструкцию `sleep`, определяющую задержку на 1 секунду в каждой итерации. Задержка в каждой итерации цикла немного замедляет скорость выполнения программы, а также обеспечивает удобный способ измерения времени, в течение которого персонаж находился на поле. Ниже в программе вы будете прибавлять 1 к переменной `rent` в каждой итерации игрового цикла, а так как односекундная задержка находится в начале цикла, это значит, что каждую секунду персонаж будет терять одно очко. Не забудьте о важности правильного оформления отступов:

```
while True:
  time.sleep(1)
```

8. Чтобы проверить, находится ли персонаж в области геозоны, нужно определить его позицию. Как и в предыдущих программах, сделать это можно с помощью `player.getTilePos()`:

```
  pos = mc.player.getTilePos()
```

9. Теперь вы подошли к самой важной части программы. Здесь, на основе четырех координат, полученных ранее, программа должна определить, находится ли персонаж на поле, и если да, отнять у него очко и вывести сообщение об этом в чат. Введите следующие инструкции:

```
if pos.x>X1 and pos.x<X2 and pos.z>Z1 and pos.z<Z2:
  rent = rent+1
  mc.postToChat("You owe rent:"+str(rent))
```

В этой точке программы будьте особенно внимательны к оформлению отступов. По отступам Python определяет принадлежность одних инструкций к другим. Обратите внимание на то, что все инструкции ниже `while True:` смещены вправо на один отступ; благодаря этому Python знает, что они принадлежат циклу `while`. Но взгляните внимательнее на инструкцию `if`: следующие за ней две строки в программе смещены вправо на два отступа. Это значит, что две инструкции являются частью инструкции `if` и будут выполняться, только если персонаж находится на поле.

Сохраните программу еще раз и запустите ее, выбрав в меню пункт Run ▸ Run Module (Запустить ▸ Запустить модуль).

Поиграйте немного, заводя персонажа на поле и выводя его оттуда. Когда персонаж находится на поле, каждую секунду в чате должно появляться сообщение с общей суммой отнятых очков. После вывода персонажа с поля сообщения должны прекратиться. Ваша программа запоминает общее количество очков, отнятых у персонажа, и когда он туда возвращается, продолжает счет потерь с числа, сохранившегося с момента последнего выхода с поля.

Чтобы сделать игру более интересной, персонаж должен что-то делать, находясь на поле. Создайте несколько случайных блоков в разных местах поля, затем снова запустите свою программу. Задание заключается в том, чтобы собрать все блоки, потеряв как можно меньше очков. Простое нажатие на блок и удаление можно расценивать как его взятие. Программа не будет знать о происходящем на поле, но это разнообразит игру. Вы можете разбросать по полю разные объекты и поставить перед друзьями задачу собрать их, а победителем объявить того, кто соберет все предметы, потеряв меньше очков!

Перемещение персонажа

Есть возможность сделать игру более сложной и интересной, если привлечь еще одну возможность Minecraft API, которая может оказаться полезной при создании собственных игр, — переместить персонажа в другую точку мира Minecraft! Чтобы проверить эту возможность, изменим имеющуюся программу *rent.py* так, чтобы при нахождении персонажа на поле более 3 секунд он катапультировался вверх и затем мог вернуться на поле.

Сначала нужно определить позицию за пределами поля, куда будет приземляться персонаж после катапультирования с поля. Проще всего это сделать, добавив по 2 к наибольшим координатам *x* и *z* поля. Для пущего эффекта нужно выбрать координату *y*, то есть высоту. Фактическое значение для выбора зависит от того, как далеко вы забрались в мир Minecraft, чтобы построить поле. Однако если оно имеет координату `y = 0`, можете выбрать значение, близкое к 10. Это будет высота, на которую катапультируется персонаж, а сила притяжения заставит его упасть обратно.

Теперь можно приступить к изменению программы *rent.py* и добавить в нее катапультирование персонажа, если он находится на поле дольше 3 секунд:

1. Добавьте в начало программы три новые константы, определяющие позицию точки в небе. Эта позиция должна находиться за пределами поля; при катапультировании персонаж будет перемещаться в нее. Новые строки в листинге ниже выделены жирно:

```
X1 = 10
Z1 = 10
X2 = 20
Z2 = 20

HOME_X = X2 + 2
HOME_Y = 10
HOME_Z = Z2 + 2
```

2. Далее нам нужно организовать подсчет времени нахождения персонажа на поле. Для этого добавьте переменную с именем `inField`. Она будет хранить число секунд, прошедших с того момента, как персонаж оказался на поле, и использовать для определения, не слишком ли много времени он там провел. И снова добавляем строку, которая ниже выделена жирно:

```
rent = 0
inField = 0
```

3. Добавьте инструкцию, прибавляющую единицу к переменной `inField`, если персонаж находится на поле. Для этого добавьте строку, выделенную жирным:

```
  if pos.x>X1 and pos.x<X2 and pos.z>Z1 and pos.z<Z2:
    rent = rent+1
    mc.postToChat("You owe rent:"+str(rent))
    inField = inField+1
```

4. Теперь нужно добавить инструкцию `else`, чтобы программа могла сбросить значение таймера в ноль, если персонаж находится за пределами поля. С особым вниманием отнеситесь к оформлению отступов в инструкции `else`, чтобы Python понимал, что вы имеете в виду. Подробнее об инструкции `else`, а также о символе `#` рассказано чуть ниже, пока просто введите код, выделенный жирно:

```
    mc.postToChat("You owe rent:"+str(rent))
    inField = inField+1
  else: # за пределами поля
    inField = 0
```

5. Добавьте в конец программы несколько строк кода, которые будут катапультировать персонажа в заданную позицию в небе, если он находится на поле дольше 3 секунд. Сила

притяжения вернет его на землю, и он возвратится на поле. Эти строки находятся в самом конце программы. Инструкция `if` должна быть смещена вправо на один отступ, потому что является частью цикла `while True:`, а инструкции, принадлежащие `if`, должны быть смещены вправо на два отступа:

```
if inField>3:
  mc.postToChat("Too slow!")
  mc.player.setPos(HOME_X, HOME_Y, HOME_Z)
```

6. Сохраните программу и запустите ее, выбрав в меню редактора пункт Run ▸ Run Module (Запустить ▸ Запустить модуль).

Теперь игра должна доставлять удовольствие, потому что вынуждает более тщательно планировать выходы в поле, чтобы избежать катапультирования! Разбросайте, как делали прежде, несколько предметов по полю и поставьте задачу себе и своим друзьям «собрать» (то есть уничтожить) максимальное количество блоков с наименьшей потерей очков.

Рис. 2.10. Персонаж катапультируется вверх, если находится на поле слишком долго

УГЛУБЛЯЕМСЯ В КОД

В этой программе вы использовали две новые для вас особенности языка Python, требующие дополнительных пояснений.

Первая — инструкция `else`. Инструкция `else` является необязательной частью инструкции `if`. Поэтому она всегда должна иметь столько же отступов, сколько предусмотрено для `if`, а инструкции, которые должны выполняться как часть `else`, должны получить дополнительный отступ.

Например:

```
if a>3:
  print("big")
else:
  print("small")
```

Если в переменной `a` хранится значение большее `3`, Python выведет в окне интерактивной оболочки слово «big» (большое), *иначе* (`else`, то есть если оно не больше `3`) он выведет слово «small» (маленькое). Иными словами, если результатом `if a>3` будет `True`, Python выведет `big`, иначе, если результатом `if a>3` будет `False`, он выведет `small`.

Инструкция `if` не обязана иметь часть `else` (она потому и называется необязательной), но иногда в программах необходимо отдельно обработать обе ситуации: когда условное выражение истинно или ложно.

В этой программе вы также впервые встретились с комментарием. Комментарии — это удобная возможность делать пометки в программах для себя или других (тех, кто будет читать ваши программы).

Комментарии начинаются с символа решетки:

```
# это комментарий
```

Комментарий может начинаться в любом месте в строке. Все, что находится за символом решетки, вплоть до конца строки, интерпретатор Python будет игнорировать.

Дополнительные приключения с наблюдением за перемещениями персонажа

В этом Приключении вы познакомились и применили на практике игровой цикл, использовали приемы определения координат и установки геозон для создания небольшой игры. Создав игру, предложите поиграть в нее своим друзьям и родственникам, попросите их сказать, что понравилось, а что нет. Эти отзывы помогут вам улучшить игру.

- Программа «Welcome Home» имеет проблему: если персонаж стоит на коврике не двигаясь, программа продолжает раз за разом выводить приветствие «welcome home» (добро пожаловать домой) и заполняет им весь чат. Подумайте, как изменить программу, чтобы она выводила приветствие только один раз, когда персонаж приходит домой?
- Отличным источником идей для создания захватывающих игр могут стать другие игры. Играйте в них и выясняйте, чем они завоевывают внимание игрока. А по ходу составляйте список: что сделать, чтобы изменить позицию персонажа.
- Поищите в интернете примеры использования геозон и посмотрите, как их можно использовать в реальной жизни. Если вам встретятся интересные идеи, возьмите их на вооружение и напишите программу для Minecraft, которая позволит воплотить эти идеи в вашу игру, основанную на том, что вы узнали в данном Приключении.

Краткая справочная таблица	
Команда	**Описание**
`import mcpi.minecraft as minecraft`	Импортирование Minecraft API
`mc = minecraft.Minecraft.create()`	Подключение к игре Minecraft

`pos = mc.player.getTilePos()`
`x = pos.x`
`y = pos.y`
`z = pos.z`
	Определение позиции персонажа

`mc.postToChat("Hello Minecraft")`	Вывод сообщения в чат Minecraft

`x = 5`
`y = 3`
`z = 7`
`mc.player.setTilePos(x, y, z)`
	Перенос персонажа в новую позицию

Новое достижение: создатель захватывающей игры, которая меняет положение персонажа в мире Minecraft.

В СЛЕДУЮЩЕМ ПРИКЛЮЧЕНИИ

В Приключении 3 вы узнаете, как конструировать большие сооружения, такие как дома, с использованием блоков Minecraft и циклов Python. С помощью программ на Python строить большие сооружения намного быстрее, чем вручную. Вы сможете создавать целые города гораздо быстрее, чем кто-либо из ваших друзей.

Приключение 3
Автоматическое создание сооружений

СТРОИТЕЛЬСТВО СООРУЖЕНИЙ в Minecraft доставляет массу удовольствия. Вы можете построить почти все, что угодно, ограниченное лишь размерами мира Minecraft и воображением, — дома, замки, подземные водопроводы, многоэтажные гостиницы с плавательными бассейнами и целые города! Но для строительства сложных объектов из блоков разных типов понадобится много времени и сил, особенно при наличии множества повторяющихся участков. А если автоматизировать некоторые строительные операции? Не будет ли это настоящим волшебством для ваших друзей?

Язык программирования, такой как Python, идеально подходит для автоматизации сложных задач. В этом Приключении вы познакомитесь с рядом волшебных возможностей, которые позволят вам автоматически создать много блоков в мире Minecraft, а затем, использовав циклические инструкции, построить большие сооружения, например улицы с домами (рис. 3.1). Ваши друзья смогут бродить по созданным вашей программой улицам и поражаться тому, что вам удалось создать столь сложную конструкцию. Вы также узнаете, как вводить числа с клавиатуры, — это даст возможность писать программы, поведение которых может меняться без изменения программного кода.

Рис. 3.1. Дом, построенный из блоков Minecraft

Создание блоков

Блоки каждого типа в мире Minecraft имеют собственный номер. Если вы обычный игрок, вы наверняка знаете множество типов блоков и их номера. Однако программный интерфейс Minecraft дает возможность ссылаться на типы блоков по именам, а не по числам, что облегчает работу с ними в программах. Имена — лишь константы (с ними вы уже встречались в Приключении 2), которые хранят номера типов блоков. Полный список наиболее часто используемых типов блоков, их названия и номера см. в Приложении А.

Мир Minecraft незримо делится на кубы (каждый со своими уникальными координатами). Компьютер запоминает номер типа блока в каждом незримом кубе, даже если это лишь блок типа `AIR` (воздух). Как программист для Minecraft, вы в любой момент можете затребовать у программного интерфейса Minecraft тип блока в кубе с любыми координатами или изменить его. Это приведет к изменению блока, отображаемого на экране. Вы можете, например, строить мосты, заменяя блоки типа `AIR` (воздух) блоками типа `STONE` (камень). Именно так вы будете поступать в Приключении 4.

Теперь напишем простую программу, демонстрирующую возможность автоматического создания одиночного блока прямо перед персонажем. Научившись создавать один блок в мире Minecraft, вы без труда сможете строить что угодно!

1. Запустите игру Minecraft, IDLE и сервер (вы должны обладать достаточным практическим навыком, чтобы сделать это самостоятельно). Если что-то забыли, загляните в Приключение 1, чтобы освежить память.
2. Откройте IDLE — интегрированную среду разработки на Python. Это ваше окно в язык программирования Python, куда вы можете вводить свои программы и запускать их.
3. Выберите пункт меню File ▸ New File (Файл ▸ Создать файл). Сохраните вновь созданную программу, выбрав File ▸ Save As (Файл ▸ Сохранить как), с именем *block.py*. Не забывайте, что программы нужно сохранять в папке *MyAdventures*, иначе они не будут работать.
4. Импортируйте необходимые модули. В этой программе используется еще один дополнительный модуль с именем `block`, хранящий все константы с номерами всех типов блоков, поддерживаемых в Minecraft. Для этого введите

```
import mcpi.minecraft as minecraft
import mcpi.block as block
```

5. Подключитесь к игре Minecraft:

```
mc = minecraft.Minecraft.create()
```

6. Получите позицию персонажа и сохраните координаты в переменной `pos`. Эта позиция будет использована для вычисления координат позиции перед персонажем, где вы создадите новый блок:

```
pos = mc.player.getTilePos()
```

7. Теперь создайте блок перед персонажем, указав координаты относительно его позиции. Подробнее об **относительных координатах** рассказано после данного раздела. Исполь-

зование `pos+3` в программе гарантирует, что каменный блок не появится над персонажем! Для этого введите

```
mc.setBlock(pos.x+3, pos.y, pos.z, block.STONE.id)
```

8. Сохраните программу, выбрав в меню пункт File ▸ Save (Файл ▸ Сохранить), и запустите ее, выбрав Run ▸ Run Module (Запустить ▸ Запустить модуль).

После этого рядом с персонажем должен появиться каменный блок. (Если вы не видите его, попробуйте повернуть персонажа, чтобы увидеть блок.) Вы только что заставили мир Minecraft создать блок перед персонажем! Используя этот простой прием, вы можете автоматически создавать по-настоящему интересные строения.

Каменные блоки не самые интересные вещи в мире Minecraft. Экспериментируйте со своей программой, заменяя тип `STONE` блока другими типами. На платформе Windows поддерживаются не все типы блоков. Учтите это в своих экспериментах!

Например, тип `GLOWING_OBSIDIAN` не поддерживается в версиях Minecraft для Windows. Некоторые блоки демонстрируют интересное поведение под действием силы тяжести, например `WATER` (вода) и `SAND` (песок). Обязательно поэкспериментируйте с ними!

Относительные координаты — это координаты, описывающие местоположение относительно некоторой точки (например, относительно персонажа в мире Minecraft). Так, координаты `pos.x`, `pos.y+10`, `pos.z+3` описывают местоположение в мире Minecraft, отстоящее на 10 блоков выше и на 3 блока южнее персонажа: иными словами, местоположение относительно координат персонажа. При перемещении персонажа в мире Minecraft будут меняться и относительные координаты.

Абсолютные координаты — это координаты фиксированного местоположения, представляющие некоторую точку (например, блок в мире Minecraft). Примером абсолютных координат могут служить координаты `x=10`, `y=10`, `z=15`. Всякий раз, ссылаясь на них, вы будете получать один и тот же блок с координатами `10`, `10`, `15`.

Если вы уже преодолели Приключение 2, загрузите и запустите программу *whereAmI.py* и перейдите в мире Minecraft туда, где есть немного свободного пространства. Запишите координаты x, y и z местоположения игрока. Измените программу *block.py*, указав в инструкции `setBlock()` абсолютные координаты, запустите ее и посмотрите, что получится. Как вы думаете, абсолютные и относительные координаты пригодятся вам в программах?

Создание нескольких блоков

Используя этот простой прием, вы будете строить что угодно! Теперь можно расширить предыдущую программу и создать несколько блоков. Формируя конструкции из блоков Minecraft, вам придется использовать простые арифметические вычисления — для определения координат каждого нового блока.

Сейчас вы добавите в свою программу *build.py* создание еще пяти блоков перед персонажем, используя отдельную инструкцию `setBlock()` для каждого блока. Программа создаст структуру, напоминающую точки на грани кубика для игры. Выполните следующие шаги:

1. Выберите пункт меню File ▸ Save As (Файл ▸ Сохранить как) и сохраните программу в файле с именем *dice.py*.
2. Добавьте следующие строки в конец программы:

```
mc.setBlock(pos.x+3, pos.y+2, pos.z,   block.STONE.id)
mc.setBlock(pos.x+3, pos.y+4, pos.z,   block.STONE.id)
mc.setBlock(pos.x+3, pos.y,   pos.z+4, block.STONE.id)
mc.setBlock(pos.x+3, pos.y+2, pos.z+4, block.STONE.id)
mc.setBlock(pos.x+3, pos.y+4, pos.z+4, block.STONE.id)
```

3. Сохраните программу и запустите ее. Вы должны увидеть перед персонажем простую конструкцию — шесть точек на грани кубика для игры (рис. 3.2).

Рис. 3.2. Шесть каменных точек перед персонажем

Измените программу *dice.py* так, чтобы пространство вокруг каменных точек было заполнено белыми блоками и картина больше напоминала квадратную грань кубика для игры. Поэкспериментируйте с настройками блоков `STONE` и `AIR`, создайте грани с разным количеством точек. Помните, что подробный список блоков есть в Приложении А.

Использование циклов for

Конструирование из отдельных блоков позволит создавать все что угодно, но этот подход напоминает создание конструкций вручную, блок за блоком. Соответственно, нужен способ повторить операцию, чтобы можно было создавать большие конструкции, не увеличивая размер программы.

К счастью, как и в других языках программирования, в Python имеются управляющие конструкции, называемые циклами. Вы уже познакомились с циклами в Приключении 2, в виде игрового цикла `while True:`. Цикл — это простой способ выразить многократное выполнение повторяющихся действий на языке программирования, таком как Python.

Цикл, который будет использоваться далее, называется `for`; иногда его называют **циклом со счетчиком**, потому что он отсчитывает фиксированное число итераций.

Цикл `for` называют **циклом со счетчиком** или циклом, управляемым счетчиком, потому что он отсчитывает фиксированное число итераций. В инструкции `range()` вы определяете, сколько итераций цикла нужно выполнить. Цикл `for` в Python необычный: он может отсчитывать не только числа, обладает дополнительными возможностями, с которыми вы познакомитесь в Приключении 5.

Строительство многосоставных блоков с помощью цикла for

Чтобы понять, как работает цикл `for`, воспользуйтесь интерактивной оболочкой Python и выполните следующие шаги:

1. Щелкните на окне интерактивной оболочки Python, чуть правее последнего приглашения к вводу `>>>`.
2. Введите следующую строку и нажмите клавишу Enter на клавиатуре:

   ```
   for a in range(10):
   ```

   Python ничего не выполнит, потому что ожидает ввода следующей инструкции или инструкций, принадлежащих циклу (их часто называют телом цикла).

3. Оболочка Python автоматически добавит отступ к следующей строке, чтобы интерпретатор Python понял, что следующая ниже инструкция `print()` принадлежит циклу `for`. Теперь введите эту инструкцию `print()`:

```
  print(a)
```

Как и в случае с использованием цикла `while`, на основе которого построен игровой цикл в Приключении 2, программные инструкции, принадлежащие циклу `for`, должны иметь на один отступ больше, чтобы Python знал, какие инструкции должны повторяться в цикле.

4. Дважды нажмите клавишу Enter. Первое нажатие сообщит, что завершен ввод инструкции `print()`, второе — что завершен ввод цикла.

В результате в окне интерактивной оболочки Python должны появиться числа от 0 до 9.

УГЛУБЛЯЕМСЯ В КОД

Только что использованный цикл `for` имеет ряд интересных свойств:

```
for a in range(10):
  print(a)
```

Имя `a` в этом фрагменте называют управляющей переменной цикла, которая используется для хранения значения цикла в каждой итерации. В первой итерации она будет хранить число 0, во второй — число 1, и т. д.

Инструкция `range(10)` сообщает Python, что вы желаете получить последовательность из десяти чисел, от 0 до 9, по которым последовательно будет шагать цикл `for`.

Двоеточие (`:`) в конце — точно такое же двоеточие, как в инструкциях `while` и `if` в предыдущих приключениях. Двоеточие отмечает позицию, где начинается тело инструкции. Тело инструкции — это программный код, который (в данном случае) будет выполнен 10 раз.

Любые инструкции на языке Python внутри цикла (инструкции с дополнительным отступом относительно инструкции `for`) могут обращаться к переменной цикла `a`, и эта переменная в каждой итерации будет хранить новое значение.

Управляющую переменную цикла можно использовать в любых инструкциях в теле цикла. Переменная цикла необязательно должна называться `a`. Вы можете придумать любое другое имя; считается хорошей практикой давать переменным говорящие имена, чтобы тем, кто будет читать вашу программу, было проще ее понять. В данном случае вместо имени `a` переменной цикла можно было бы дать имя `number_of_times`.

Строительство башни с помощью цикла for

Хотелось бы вам построить высоченную башню из блоков в мире Minecraft? Теперь, когда вы познакомились с циклом `for`, можете легко это сделать. Просто выполните следующие шаги:

1. Начните писать новую программу, выбрав в меню пункт File ▸ New File (Файл ▸ Создать файл). Затем выберите File ▸ Save As (Файл ▸ Сохранить как) и сохраните программу с именем *tower.py*.
2. Как обычно, импортируйте необходимые модули:

```
import mcpi.minecraft as minecraft
import mcpi.block as block
```

3. Подключитесь к игре Minecraft:

```
mc = minecraft.Minecraft.create()
```

4. Если строить башню в месте, где находится персонаж, ее будет проще найти. Поэтому сначала определите позицию персонажа:

```
pos = mc.player.getTilePos()
```

5. Башня будет иметь высоту 50 блоков. Соответственно, начнем строительство с определения цикла `for`, выполняющего 50 итераций. Не забудьте добавить двоеточие (`:`) в конце строки!

```
for a in range(50):
```

6. Следующая строка должна иметь отступ как принадлежащая телу цикла `for`. Высотой в мире Minecraft управляет координата *y*, поэтому добавьте переменную цикла `a` к координате *y* персонажа. Благодаря этому в каждой итерации новый блок будет создаваться на большей высоте:

```
mc.setBlock(pos.x+3, pos.y+a, pos.z, block.STONE.id)
```

7. Сохраните программу и запустите ее. Все получилось? Ваш цикл `for` должен создать массивную башню перед персонажем, как показано на рис. 3.3. Подсчитайте: действительно ли она высотой 50 блоков?

В предыдущем примере цикл `for` выполняет 50 итераций, и в каждой есть строка с дополнительным отступом. При этом в каждой итерации переменная `a` имеет значение на единицу больше, чем в предыдущей итерации. Она складывает значение переменной `a` (управляющей переменной цикла) с переменной `pos.y` (хранит высоту персонажа). В результате получается башня высотой 50 блоков!

Рис. 3.3. Эта высоченная башня была создана в мире Minecraft с помощью цикла for

Измените инструкцию `range()` в цикле `for`, чтобы создать еще более высокую башню. Насколько высокой должна быть башня, чтобы ее вершину не было видно?

Очистка пространства

Иногда в игре Minecraft сложно найти свободное пространство, достаточное для строительства больших сооружений. Обычно персонажа окружают горы, покрытые деревьями, из-за чего свободного места для строительства чего-то крупного оказывается мало. Вы можете решить эту проблему, написав программу, которая будет очищать некоторое пространство для строительства.

Использование setBlocks для ускорения строительства

Все блоки в мире Minecraft имеют идентификационные числа (`id`), в том числе блоки с пустым пространством, которые вы видите перед собой. Они имеют идентификационное число `block.AIR.id`. То есть если всем блокам на большом пространстве присвоить число `block.AIR.id`, можно освободить достаточно пространства для строительства других объектов. Идентификационное число `block.AIR.id` равно `0`, а идентификационное число `block.`

`STONE.id` — `1`. Не забывайте, что пустое пространство в мире Minecraft в действительности заполнено блоками с идентификационными номерами `block.AIR.id` , то есть кубами с ребром в 1 метр, заполненными воздухом.

Компьютеры работают очень быстро, но чем больше программных инструкций будет выполняться в цикле `for`, тем медленнее будет работать программа. Чтобы проанализировать ваш запрос на установку блока и изменение его типа и отобразить все на экране, игре Minecraft требуется время. Вы могли бы всегда использовать циклы, записывающие в сотни блоков перед персонажем идентификационное число `block.AIR.id`, чтобы очистить пространство, и этот прием будет работать, но очень медленно! К счастью, существует более быстрый способ установки множества блоков за один присест — инструкция `setBlocks()`. (Обратите внимание на дополнительную букву `s` в конце имени.) `setBlock()` устанавливает один блок, а `setBlocks()` — сразу много блоков!

Программный интерфейс Minecraft имеет инструкцию `setBlocks()`, которую можно использовать, чтобы присвоить всем блокам в трехмерной области одно и то же идентификационное число. Поскольку эта инструкция выполняет единственный запрос к игре, Minecraft может оптимизировать свою работу и выполнить задание намного быстрее, чем при использовании `setBlock()` внутри цикла `for`, как это делалось во время строительства башни.

Так как инструкция `setBlocks()` работает с трехмерными областями, ей нужно передать два набора координат: координаты одного угла прямоугольной области и противоположного угла. Координаты в трехмерном пространстве определяются тремя числами, поэтому, чтобы описать трехмерную область в мире Minecraft, нужно шесть чисел.

Теперь напишите небольшую вспомогательную программу, которую можно в любой момент использовать, чтобы очистить пространство для строительства новых сооружений:

1. Создайте новую программу, выбрав в меню пункт File ▸ New File (Файл ▸ Создать файл).
2. Сохраните программу, выбрав в меню пункт File ▸ Save As (Файл ▸ Сохранить как), с именем *clearSpace.py*.
3. Импортируйте необходимые модули:

```
import mcpi.minecraft as minecraft
import mcpi.block as block
```

4. Подключитесь к игре Minecraft:

```
mc = minecraft.Minecraft.create()
```

5. Вам требуется очистить пространство перед персонажем, поэтому используйте относительные координаты. Для начала получите координаты персонажа:

```
pos = mc.player.getTilePos()
```

6. Теперь очистите пространство размером 50 × 50 × 50 блоков. Левый нижний угол области совпадает с позицией персонажа, а правый верхний будет иметь координаты *x*, *y* и *z* на 50 больше:

```
mc.setBlocks(pos.x, pos.y, pos.z, pos.x+50, pos.y+50, ↩
 pos.z+50, block.AIR.id)
```

7. Сохраните программу.

Самое фантастическое в этой программе то, что она позволяет перейти в любую точку мира Minecraft и очистить область размером 50 × 50 × 50. Просто запустите ее с помощь пункта меню Run ▸ Run Module (Запустить ▸ Запустить модуль), и все деревья и горы — то, что окажется перед вашими глазами, — исчезнет как по волшебству!

Побродите по миру Minecraft и попробуйте очистить больше пространства.

Чтение ввода с клавиатуры

В программу *clearSpace.py* можно добавить дополнительное удобство, которое позволит менять размеры очищаемого пространства. То есть если вам требуется построить небольшое здание, вы можете очистить небольшой объем, а если собираетесь построить большое сооружение, можете очистить сразу огромный объем пространства.

Для этого можно воспользоваться новой для вас инструкцией `input()`, которая читает число с клавиатуры. С улучшенной программой вы сможете перейти в любую точку мира Minecraft, запустить программу и просто ввести число, определяющее размер области для очистки, после чего программа очистит требуемый объем. Это значит, что вам не придется менять программу для каждой области очистки нового размера.

Существует две основные версии Python: Python 2 и Python 3. Программный интерфейс Minecraft теперь работает в обеих версиях. Однако Python 2 использует инструкцию `raw_input()` для чтения с клавиатуры, тогда как Python 3 — `input()`. Все программы в этой книге используют Python 3; если вы видите в интернете программы, предназначенные для использования с Python 2, то увидите `raw_input()`.

Теперь, когда вы знаете, что делает инструкция `input()`, добавим ее в программу. Для этого достаточно внести в *clearSpace.py* два небольших изменения:

1. Сначала добавьте `input()`, чтобы попросить пользователя ввести число, которое будет определять размер области пространства для очистки, и сохраните это число в переменной с именем `size`. Как в Приключении 2 использовалась инструкция `str()` для преобразования значений в строку, так здесь используется инструкция `int()` для преобразования строки, возвращаемой инструкцией `input()`, в число, которое затем мож-

но использовать в вычислениях. Попробуйте убрать инструкцию `int()` и посмотрите, что произойдет! Добавьте в программу только новые строки, выделенные **жирным шрифтом**:

```
pos = mc.player.getTilePos()
size = int(input("size of area to clear? "))
```

2. Измените строку с инструкцией `setBlocks()` так, чтобы вместо числа 50 в ней использовалась переменная `size`:

```
mc.setBlocks(pos.x, pos.y, pos.z,
             pos.x + size, pos.y + size, pos.z + size,
             block.AIR.id)
```

3. Сохраните программу и запустите ее, выбрав в меню пункт Run ▸ Run Module (Запустить ▸ Запустить модуль).

Теперь перейдите в некоторую точку мира Minecraft, где есть множество гор и деревьев. Введите число в интерактивной оболочке Python — например, 100 или любое другое — и нажмите клавишу Enter на клавиатуре. Все деревья и горы рядом с персонажем исчезнут, как показано на рис. 3.4, и вы получите достаточно свободного пространства для строительства.

Сохраните эту маленькую полезную программу; она еще не раз пригодится, когда вы захотите очистить немного места в мире Minecraft.

Рис. 3.4. Обратите внимание на то, что после выполнения clearSpace.py некоторые деревья оказались распиленными вдоль ствола!

Программа *clearSpace.py* очищает пространство в форме куба, используя позицию персонажа как левый нижний угол этого пространства. Однако в зависимости от того, куда персонаж повернут лицом, вы можете не увидеть очищенное пространство, пока не повернетесь вокруг. Было бы гораздо удобнее, если бы очистка пространства осуществлялась вокруг персонажа. Для этого можно изменить вычисление относительных координат углов очищаемого пространства в инструкции `setBlocks()` так, чтобы персонаж оказывался в центре куба. Еще можно было бы заставить *clearSpace.py* «рыть котлованы», уменьшив координату *y*, чтобы освободить немного пространства для фундамента, газонов и других наземных и подземных сооружений. Попробуйте сделать это прямо сейчас и посмотрите, что получится. Эта программа будет намного полезней!

Строительство дома

Играя в Minecraft в режиме выживания, первое, что необходимо сделать, — построить убежище для вашего персонажа, чтобы защитить его от опасностей ночного мира. Было бы здорово, если бы такое убежище можно было построить нажатием одной клавиши. К счастью, благодаря поддержке программирования для Minecraft, сложные задачи удается решать именно таким способом — нажатием одной клавиши.

В программе *clearSpace.py* вы узнали, как большому количеству блоков присвоить одно идентификационное число, определяющее тип блока, использовав для этого единственную программную инструкцию. Теперь вам предстоит узнать, как с помощью того же приема построить дом.

Один из способов — строить каждую стену отдельной инструкцией `setBlocks()`. Для этого потребуется использовать четыре отдельные инструкции `setBlocks()` и произвести множество арифметических вычислений, чтобы определить координаты всех углов во всех стенах. К счастью, есть более простой вариант строительства прямоугольных сооружений: достаточно построить большой куб и затем с помощью `setBlocks()` «вырезать» в нем помещения, заменив внутренние блоки блоками с типом `AIR`.

Прежде чем двинуться дальше, потратим немного времени на арифметические вычисления, которые нужно выполнить при строительстве здания, чтобы получить координаты углов внутренних помещений. На рис. 3.5 показан чертеж дома, где отмечены все координаты, которые потребуется вычислить. Строительство сложного сооружения следует начинать с чертежа на бумаге, имеющего все координаты. Это необычный дом: размеры и позиции его окон и дверей вычисляет программа. Почему это важно, описано ниже.

На сайте книги можно найти видеоурок по строительству дома. Посетите веб-сайт www.wiley.com/go/adventuresinminecraft2e и выберите видеоурок для Приключения 3.

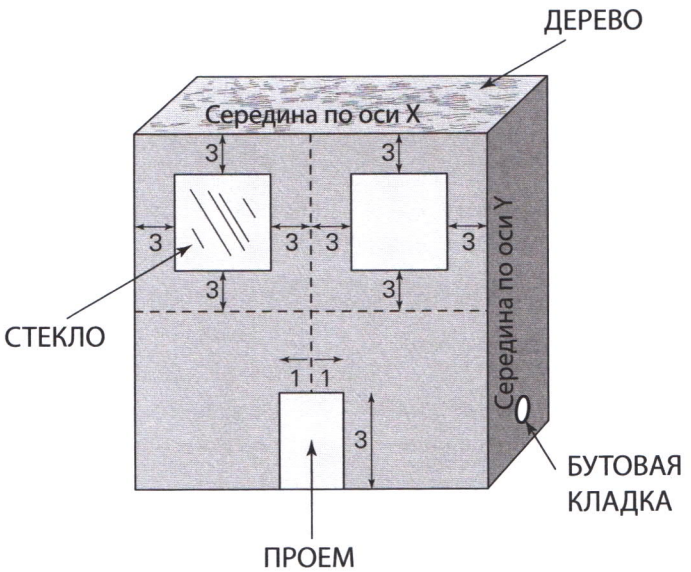


Рис. 3.5. Проект дома на бумаге со всеми важными координатами на нем

Теперь, когда у вас есть план дома, попробуйте его построить, выполнив следующие шаги:

1. Создайте новую программу, выбрав в меню пункт File ▸ New File (Файл ▸ Создать файл).
2. Сохраните программу, выбрав в меню пункт File ▸ Save As (Файл ▸ Сохранить как), с именем *buildHouse.py*.
3. Импортируйте необходимые модули:

```
import mcpi.minecraft as minecraft
import mcpi.block as block
```

4. Подключитесь к игре Minecraft:

```
mc = minecraft.Minecraft.create()
```

5. Определите константу с размером дома. Константа `SIZE` будет часто фигурировать в процессе строительства. Использование константы вместо числа 20 упростит изменение размеров дома:

```
SIZE = 20
```

Помните, что в соответствии с соглашениями, принятыми программистами, имена констант записываются только прописными буквами. В языке Python нет требований, но данный способ именования поможет вам отличить константы от переменных и избежать их изменения во время работы программы.

6. Определите позицию персонажа, чтобы получить координаты, рядом с которыми будет создаваться дом:

```
pos = mc.player.getTilePos()
```

7. Сохраните координаты *x*, *y* и *z* в новых переменных. Это поможет упростить ввод и чтение следующих нескольких программных инструкций и потом, когда вы будете решать сложные задачи по конструированию здания. Переменная *x* должна содержать координату, отстоящую на 2 от координаты персонажа, чтобы дом не оказался над его головой:

```
x = pos.x+2
y = pos.y
z = pos.z
```

8. Вычислите две переменные с именами `midx` и `midy`, отмечающие середину фасада дома по осям *x* и *y*. Это упростит вычисление координат для окон и дверного проема; если потом размеры дома изменятся, окна и двери изменят свои позиции соответственно:

```
midx = x + SIZE/2
midy = y + SIZE/2
```

9. Постройте внешние стены блока как большой куб. Для этого можно выбрать блоки любого типа, но нам кажется, что бутовая кладка из булыжников подойдет лучше всего и сделает дом похожим на старинный:

```
mc.setBlocks(x, y, z,
             x + SIZE, y + SIZE, z + SIZE,
             block.COBBLESTONE.id)
```

10. Теперь вырежьте внутреннее пространство дома, заполнив его воздухом. Обратите внимание на использование простых арифметических операций для вычисления координат блоков с воздухом внутри дома относительно координат внешних углов каменных стен:

```
mc.setBlocks(x + 1, y, z + 1,
             x + SIZE - 1, y + SIZE - 1, z + SIZE - 1,
             block.AIR.id)
```

11. Вырежьте дверной проем, снова использовав блоки типа `AIR`. Вы не будете использовать здесь обычную дверь Minecraft, потому что дом достаточно велик. Вместо этого создайте большой дверной проем высотой в три блока и шириной в два. Проем должен размещаться в середине фасада, поэтому используйте `midx` для определения координаты *x* в середине стены:

```
mc.setBlocks(midx-1, y, z,
             midx + 1, y + 3, z,
             block.AIR.id)
```

12. Вырежьте два окна, использовав блоки типа `GLASS` (стекло). Так как дом получился большим, окна должны начинаться через три блока от внешнего края стены и через три блока от средней линии фасада. Если вы измените константу `SIZE` и снова запустите программу, вычисления автоматически изменят положение окон и дверного проема, и дом сохранит привлекательный внешний вид.

```
mc.setBlocks(x + 3, y + SIZ - 3, z,
             midx - 3, middy + 3, z,
             block.GLASS.id)
```

```
mc.setBlocks(midx + 3, y + SIZE - 3, z,
             x + SIZE - 3, middy + 3, z,
             block.GLASS.id)
```

13. Добавьте деревянную крышу:

```
mc.setBlocks(x, y + SIZE - 1, z,
             x + SIZE, y + SIZE - 1, z + SIZE,
             block.WOOD.id)
```

14. Добавьте ковер из шерсти:

```
mc.setBlocks(x + 1, y - 1, z + 1,
            x + SIZE - 2, y - 1, z + SIZE - 2,
            block.WOOL.id, 14)
```

В конце этой инструкции `setBlocks()`обратите внимание на дополнительное число. Оно определяет цвет ковра: 14 означает красный цвет. Подробнее об этом будет рассказано ниже, во врезке «Углубляемся в код».

Сохраните программу, затем перейдите в точку мира Minecraft, где достаточно места для постройки дома, и запустите программу, выбрав в меню пункт Run ▸ Run Module (Запустить ▸ Запустить модуль). Вы должны увидеть, как ваш дом возникает будто по волшебству! Зайдите внутрь и исследуйте его. Посмотрите на потолок и улицу сквозь окна, проникнитесь тем, с какой скоростью был построен дом (рис. 3.6).

Не забудьте, что программа *buildHouse.py* всегда строит дом рядом с персонажем. Побродите по миру Minecraft и запустите *buildHouse.py* еще раз, чтобы построить еще один дом. Теперь вы можете застроить домами всю округу. Круто, правда?

Рис. 3.6. Дом, построенный программой на Python

УГЛУБЛЯЕМСЯ В КОД

При создании ковра в инструкции `setBlocks()` использовалось дополнительное число:

```
mc.setBlocks(x1, y1, z1, x2, y2, z2, block.WOOL.id, 14)
```

Давайте посмотрим, что оно означает.

`WOOL` (шерсть) — один из самых интересных типов блоков, потому что при их создании допускается передавать то, что называется «дополнительными данными». Вы можете не только создать блок из шерсти (WOOL), но и определить его дополнительные свойства. В мире Minecraft существуют и другие типы блоков с аналогичными возможностями, но для каждого типа дополнительные данные имеют разный смысл. Для блоков типа `WOOL` в качестве дополнительных данных передается цвет. Эта полезная особенность придает блокам `WOOL` особую гибкость, потому что позволяет выбирать цвет из палитры поддерживаемых цветов и создавать более реалистичные, красочные орнаменты.

В конце этого Приключения представлена таблица со списком всех номеров и соответствующих им цветов для блоков `WOOL`.

Если вам доводилось видеть в интернете другие программы для Minecraft, у вас наверняка возник вопрос: почему в этой книге типы блоков обозначаются `block.WOOL.id`? Почему за названием типа блока всегда следует `.id`? И почему в других программах такое окончание отсутствует? Это объясняется тем, что иногда `setBlock()` и `setBlocks()` не работают, получив дополнительные данные, если не указать `.id` в конце типа блока. Поэтому для единообразия и простоты мы решили обязательно использовать окончание `.id`, чтобы все инструкции выглядели похожими. Окончание `.id` требуется не всегда, но его присутствие не мешает, поэтому есть смысл постоянно его добавлять и не запоминать, в каких случаях окончание необходимо, а в каких — нет.

Попробуйте изменить константу `SIZE` в своей программе, подставив большое число, например 50, и запустите программу. Как изменился вид дома? Попробуйте присвоить константе `SIZE` маленькое число, например 10. Как изменился дом теперь? На ваш взгляд, что произойдет, если в константе `SIZE` сохранить очень маленькое число?

Строительство нескольких домов

Строительство одного здания программным способом вызывает восхищение, но зачем останавливаться на достигнутом? Вспомните программу *tower.py*, написанную ранее в этом При-

ключении: было бы совсем несложно написать цикл `for`, повторяющий программные инструкции фиксированное число раз. То есть таким способом можно построить целую улицу с домами, и даже целый город, многократно повторяя в цикле программу постройки.

Однако прежде, чем заняться этим, давайте усовершенствуем программу постройки дома, чтобы потом с ней было проще справиться.

Использование функций в языке Python

Позднее вам наверняка захочется построить целый город с домами разной архитектуры. Программа такого строительства может стать очень большой и сложной, но, к счастью, в языке Python есть возможность упаковать сложности в небольшие фрагменты программного кода, пригодные для многократного использования. Такие фрагменты называют **функциями**.

Функции в языке Python позволяют группировать программные инструкции и присваивать этим группам имена. Всякий раз, когда понадобится выполнить такую группу инструкций, достаточно использовать ее имя с круглыми скобками после него.

Возможно, вы не заметили, но мы используем функции повсюду в этой книге, начиная с того момента, как в самом первом приключении отправили сообщение в чат Minecraft с помощью `mc.postToChat("hello")`. Здесь `postToChat()` — функция, которая является лишь группой программных инструкций, выполняющихся всякий раз, когда в программе используется слово `postToChat()`.

Прежде чем изменить программу *buildHouse.py* и задействовать в ней функцию для строительства дома, попробуем определить какую-нибудь функцию в интерактивной оболочке Python, чтобы понять основную идею:

1. Щелкните на окне интерактивной оболочки Python, чтобы вывести его на передний план.
2. Введите следующую строку, определяющую новую функцию с именем `myname`:

   ```
   def myname():
   ```

   Инструкция `def` означает «определить (define) новую функцию с указанным именем». Подобно инструкциям `while`, `if` и `for`, строка с определением имени функции должна заканчиваться двоеточием (`:`), чтобы Python понял: инструкции, следующие дальше, образуют тело функции.
3. Интерактивная оболочка автоматически добавит отступ в следующей строке, чтобы интерпретатор Python понял, что она (строка) является частью функции. Введите не-

сколько строк с инструкциями `print`, которые выводят ваше имя и некоторые сведения о вас. Не удивляйтесь, что ничего не происходит по мере ввода этих строк — все идет своим чередом, как должно быть. Наберитесь терпения, все станет понятно через минуту!

```
print("my name is David")
print("I am a computer programmer")
print("I love Minecraft programming")
```

4. Дважды нажмите клавишу Enter, и интерактивная оболочка Python поймет, что ввод инструкций с отступом завершен.
5. Запросите оболочку Python выполнить эту функцию, для чего введите ее имя с круглыми скобками после него:

```
myname()
```

6. Попробуйте ввести `myname()` еще несколько раз и посмотрите, что произойдет.

На первый взгляд кажется странным, что при вводе инструкций в интерактивной оболочке Python ничего не происходит. Обычно, когда вводится инструкция, сразу после нажатия на клавишу Enter что-то случается. Почему на этот раз все иначе? Дело в том, что вы не просто вводили инструкции для выполнения, а «определяли» (define) новую функцию с именем `myname` и требовали от Python запомнить три инструкции `print` как принадлежащие ей. Подчиняясь вашему требованию, Python вносил эти инструкции в память компьютера, вместо того чтобы немедленно их выполнять. Теперь всякий раз, как вы будете вводить команду `myname()`, он будет выполнять эти сохраненные инструкции и выводить три строки текста на экран.

Функции — очень мощный инструмент. Они позволяют связать любое число программных инструкций с простым именем, чем-то напоминая мини-программы. Всякий раз, когда понадобится выполнить эти инструкции, достаточно ввести имя функции.

Давайте воспользуемся замечательной возможностью и определим функцию строительства дома. Потом, когда вам понадобится построить дом из камня, будет достаточно ввести команду `house()`, и она автоматически построит дом!

1. Чтобы не уничтожать рабочую программу *buildHouse.py*, выберите в меню редактора пункт File ▸ Save As (Файл ▸ Сохранить как) и сохраните программу с новым именем *buildHouse2.py*.
2. В самом начале программы, ниже инструкций `import`, определите новую функцию с именем `house`:

```
def house():
```

3. Теперь добавьте отступы ко всем строкам, начиная с определений переменных `midx`, `midy` и инструкций `setBlocks()`, чтобы они оказались принадлежащими инструкции

`def house():`. Ниже показано, как должна выглядеть программа. Будьте внимательны к оформлению отступов:

```
import mcpi.minecraft as minecraft
import mcpi.block as block
mc = minecraft.Minecraft.create()
SIZE = 20
def house():
  midx = x + SIZE/2
  midy = y + SIZE/2
  mc.setBlocks(x, y, z, x+SIZE, y+SIZE, z+SIZE, ↩
    block.COBBLESTONE.id)
  mc.setBlocks(x+1, y+1, z+1, x+SIZE-2, y+SIZE-2, ↩
    z+SIZE-2, block.AIR.id)
  mc.setBlocks(x+3, y+SIZE-3, z, midx-3, midy+3, z, ↩
    block.GLASS.id)
  mc.setBlocks(midx+3, y+SIZE-3, z, x+SIZE-3, midy-3, z, ↩
    block.GLASS.id)
  mc.setBlocks(x, y+SIZE, z, x+SIZE, y+SIZE, z+SIZE, ↩
    block.SLATE.id)
  mc.setBlocks(x+1, y+1, z+1, x+SIZE-1, y+1, z+SIZE-1, ↩
    block.WOOL.id, 7)
pos = mc.player.getTilePos()
x = pos.x
y = pos.y
z = pos.z
house()
```

4. Обратите внимание на последнюю инструкцию в программе: в ней просто указано имя функции `house()`. Эта строка запускает программные инструкции, хранящиеся в памяти компьютера, которые были записаны туда инструкцией `def house():`.
5. Сохраните программу, перейдите в новую точку и запустите программу. Перед вами должен появиться новый дом.

Вы можете подумать: «Ну и что? Я изменил программу, переместив несколько инструкций с места на место, но она делает то же самое!» Иногда при создании компьютерных программ необходимо изменить их организацию, чтобы потом творить нечто еще более удивительное. Именно это вы сейчас и проделали — реорганизовали программу. Зато теперь вы можете многократно использовать программный код, строящий дом.

Функция `house()` всегда строит дом относительно координат, хранящихся в переменных *x*, *y* и *z*. Добавьте в конец программы еще несколько инструкций, меняющих *x*, *y* и *z*, затем еще одну инструкцию `house()` — и посмотрите, что получится. Как вы думаете, сколько домов можно построить таким способом?

УГЛУБЛЯЕМСЯ В КОД

В новой программе все инструкции `setBlocks()` были помещены внутрь функции `house()`, благодаря чему перед вами открылись новые возможности.

Переменные `x`, `y`, `z` и константа `SIZE` называются **глобальными** переменными. Таковыми они считаются потому, что получили свои значения в главной программе (не среди инструкций с отступами). Глобальные переменные можно использовать повсюду в программе, включая инструкции внутри функции `house()`. Если вы провели эксперимент, предложенный выше, и построили несколько домов, изменив значения переменных `x`, `y` и `z` в главной программе, наверняка сами убедились в такой возможности, потому что эти переменные действительно глобальные и могут использоваться повсюду в программе.

В Приключении 5 вы узнаете, что глобальные переменные могут серьезно усложнить поиск и исправление ошибок в больших программах и что существует более удобный способ хранения информации для использования в разных функциях. А пока такое использование глобальных переменных вполне оправданно, и программа достаточно мала, чтобы не вызывать никаких проблем.

Глобальная переменная — это переменная, которая может использоваться повсюду в программе. Глобальными являются любые переменные (а также константы), объявленные в строках без отступов. Они могут использоваться в любом месте программы: если есть строка `a = 1` и она не имеет отступа, вы можете использовать переменную `a` в любом месте программы.

Переменные, объявленные в строках с отступами, не являются глобальными (их часто называют локальными переменными). То есть если в программе есть строка `a = 1` и она находится среди строк с отступами, принадлежащих инструкции `def` (*внутри* функции), эта переменная будет доступна только в этих строках внутри функции. Вам еще предстоит узнать дополнительные подробности о глобальных переменных. На данный момент это все, что вам нужно о них знать.

Создание улицы из домов с помощью цикла for

Теперь вы готовы объединить все знания, которые получили в этой главе, и построить длинную улицу из домов. Если бы вы взялись строить ее вручную, потребовалось бы несколько часов и вы наверняка допускали бы ошибки при строительстве, из-за чего одни дома получались бы чуть больше или чуть меньше других. Автоматизировав строительство домов и других сооружений с помощью программ для Minecraft, вы можете ускорить получение желаемого результата и убедиться, что все дома построены в точном соответствии с заданными размерами.

Чтобы построить множество домов, нужно добавить в программу цикл `for`, как показано ниже:

1. Чтобы не уничтожить предыдущую программу *buildHouse2.py*, выберите пункт меню File ▸ Save As (Файл ▸ Сохранить как) и сохраните ее с новым именем *buildStreet.py*.
2. Добавьте в конец программы цикл `for`, непосредственно перед строкой с инструкцией `house()` и строкой, меняющей координату *x* строительства. Каждый дом начинает строиться на расстоянии `SIZE` блоков от предыдущего. Новые строки в программе выделены **жирным шрифтом**:

```
for h in range(5):
  house()
  x = x + SIZE
```

Сохраните программу и перейдите в мире Minecraft на новое место, где достаточно свободного пространства; запустите программу. Как показано на рис. 3.7, у вас должна получиться длинная улица. Зайдите в каждый дом и убедитесь, что они построены без огрехов.

Рис. 3.7. Улица из пяти одинаковых домов, построенная программой на языке Python

Как бы вы изменили цикл, чтобы вместо домов рядом он строил бы их друг над другом, и в результате получилась высокая башня? Попробуйте это сделать. Назовите функцию строительства одного блока башни `tower()` и затем используйте цикл для возведения нескольких блоков башни друг над другом в мире Minecraft.

В зависимости от того, где находится персонаж перед началом строительства домов и как много вокруг элементов ландшафта, таких как деревья и горы, вы можете наблюдать интересные эффекты на некоторых домах. Одни из них будут рассекать деревья пополам, другие — врезаться в горы. Можно наблюдать даже дом, оказавшийся на поверхности моря! Попробуйте изменить функцию `house()` так, чтобы она строила дома лишь на твердом грунте.

Добавление случайных ковров

В настоящий момент у вас должна быть масса домов в мире Minecraft, и это обстоятельство само по себе удивительно. С небольшим количеством строк на языке Python вы построили столько сооружений!

Однако кто-то из вас наверняка подумал, что длинная улица из одинаковых домов выглядит скучновато. Как бы внести разнообразие и сделать дома разными?

Как вариант, можно написать несколько разных функций `house()`, таких как `cottage()` (коттедж), `townHouse()` (одноэтажный дом) и даже `maisonette()` (двухэтажный дом), и изменить программу, чтобы она использовала разные функции в разных местах.

Другой способ разнообразить постройки — изменить их части, например ковры, чтобы при каждом запуске программы выбирались разные цвета. В этом случае даже вы как программист не будете знать, какой цвет выбран, пока не посетите все дома!

Генерирование случайных чисел

Компьютеры — очень точные машины. Во многих жизненных ситуациях мы полагаемся на их предсказуемость, выполнение одной и той же последовательности операций при каждом запуске программы. Когда вы кладете 1000 рублей на счет в банке, предполагается, что в компьютерной памяти, где хранится баланс вашего счета, прибавилось именно 1000 рублей, без ошибок и погрешностей. Понятие случайности кажется чуждым точным системам.

Но есть одна область, в которой случайность играет важную роль. Это игры. Если игра будет в точности повторяться раз за разом, она быстро наскучит, потеряет свою привлекательность. Почти все компьютерные игры используют эффект случайности, чтобы создать новый ход событий и удержать ваш интерес.

К счастью, в языке Python есть встроенный модуль, генерирующий **случайные числа** и избавляющий вас от необходимости писать свой код. Вам остается воспользоваться тем, что уже существует.

Чтобы увидеть, на что похожи случайные числа, выполните следующие шаги в интерактивной оболочке Python:

1. Щелкните на окне интерактивной оболочки Python, чтобы вывести его на передний план.
2. Импортируйте модуль `random`, чтобы пользоваться функциями, возвращающими случайные числа:

```
import random
```

3. Сгенерируйте случайное число в диапазоне от 1 до 100 и выведите его:

```
print(random.randint(1,100))
```

В результате на экране должно появиться случайное число в диапазоне от 1 до 100. Введите инструкцию `print` еще раз. Какое число вы получили на этот раз?

4. Теперь воспользуйтесь циклом `for`, чтобы вывести множество случайных чисел. Не забудьте про отступ во второй строке, чтобы Python смог понять, что инструкция `print` является частью цикла `for`:

```
for n in range(50):
  print(random.randint(1,100))
```

Два числа в скобках после имени функции `randint()` определяют диапазон, в котором должны находиться случайные числа: `1` — наименьшее из возможных чисел, `100` — наибольшее.

Случайное число, как правило, генерируется в составе последовательности случайных чисел — списка, в котором не прослеживается упорядоченность.

Компьютеры — точные машины и часто генерируют не настоящие случайные числа, а так называемые псевдослучайные числа. Они могут выглядеть как часть случайной последовательности, но в них есть некоторая упорядоченность. Подробнее о случайных числах можно прочесть по адресу https://ru.wikipedia.org/wiki/Генератор_псевдослучайных_чисел, а дополнительную информацию о настоящих случайных числах почерпнуть на сайте www.random.org.

Создание разных ковров

Выше в этом Приключении вы использовали функцию `setBlocks()` с дополнительным числом, определяющим красный цвет шерстяного ковра — блока типа `WOOL`. Допустимыми значениями цвета для блоков `WOOL` являются числа в диапазоне от 0 до 15, то есть можно выбрать любой из 16 цветов. Теперь выполните следующие шаги, чтобы изменить программу строительства дома, дополнив ее возможностью выбирать случайный цвет для ковра:

1. Сохраните программу *buildStreet.py*, выбрав в меню пункт File ▸ Save As (Файл ▸ Сохранить как), с именем *buildStreet2.py*.

2. Добавьте в начало программы инструкцию `import`, чтобы получить доступ к генератору случайных чисел:

```
import random
```

3. Измените функцию `house()` так, чтобы она каждый раз генерировала случайный цвет для ковра. Поскольку блоки типа `WOOL` принимают значение цвета в виде чисел в диапазоне от 0 до 15, именно этот диапазон нужно использовать для получения случайного числа. Сохраните его в переменной `c`, чтобы инструкция создания ковра не получилась

слишком длинной и сложной для чтения. И не забудьте правильно оформлять отступы в обеих инструкциях, которые являются частью функции `house()`:

```
c = random.randint(0, 15)
mc.setBlocks(x+1, y+1, z+1, x+SIZE-1, y+1, z+SIZE-1, ↩
block.WOOL.id, c)
```

4. Сохраните программу.

Теперь нужно найти в мире Minecraft свободное место для строительства домов. Походите вокруг, найдите подходящее место и запустите новую программу. Она должна построить еще одну улицу из домов, но на этот раз, заглянув в них, вы должны увидеть внутри ковры разного цвета. Посмотрите на рис. 3.8, чтобы знать, как должны выглядеть вновь построенные дома. Для этого вам придется зайти в каждый дом и убедиться, что ковры там действительно разного цвета!

Рис. 3.8. В каждом доме цвет ковра был выбран случайно

Определите функцию `house2()`, которая строит дом другого типа. Используйте случайные числа в главном игровом цикле `for`, чтобы случайным образом выбирать одну из двух функций строительства домов. Каждый построенный дом будет случайным. Измените программу так, чтобы она строила дома трех разных типов: чем больше типов домов строит программа, тем интереснее будет смотреться улица. Поэкспериментируйте с отрицательными координатами, такими как `y=-5`, чтобы обеспечить постройку фундаментов или даже плавательных бассейнов в домах!

Разнообразие сооружений, которые вы можете построить в мире Minecraft, в значительной степени ограничивается лишь воображением. Да, мир Minecraft имеет ограниченную высоту, из-за чего вы не можете строить очень высокие здания. Кроме того, некоторые дома, существующие в реальном мире, имеют наклонные или изогнутые очертания. Посмотрите на архитектуру небоскреба Shard («Осколок стекла»), самого высокого в Евросоюзе[1]. Такие восхитительные здания тоже можно построить в мире Minecraft, но из-за необычной архитектуры это будет не просто.

Не торопитесь, начните с простых сооружений, имеющих обычную прямоугольную форму. В Приключении 6 Мартин познакомит вас с некоторыми вспомогательными функциями, которые позволят строить здания с наклонными и изогнутыми стенами. После этого не останется ограничений по типам зданий, которые вы можете строить в мире Minecraft!

Существует еще два ограничения, не дающие разыграться бурной фантазии в мире Minecraft: объем компьютерной памяти и максимальное расстояние, на которое простирается взгляд. Большие сооружения занимают много места на компьютере, а мир Minecraft имеет границы, поскольку хранится в его памяти. Кроме того, ваш персонаж может видеть не дальше, чем на определенное и достаточно ограниченное расстояние.

Краткая справочная таблица

Команда	Описание
`import mcpi.block as block` `b = block.DIRT.id`	Импортирование и использование констант, имен блоков
`mc.setBlock(5, 3, 2, block.DIRT.id)`	Установка/изменение блока в определенной позиции

`block.AIR.id`
`block.STONE.id`
`block.COBBLESTONE.id`
`block.GLASS.id`
`block.WOOD.id`
`block.WOOL.id`
`block.SLATE.id`
	Некоторые типы блоков для строительства домов

`mc.setBlocks(0,0,0,5,5,5,block.DIRT.id)`	Установка/изменение множества блоков сразу

[1] https://ru.wikipedia.org/wiki/The_Shard. — *Примеч. пер.*

Все новые инструкции Python и Minecraft, которые вы изучили в этой главе, перечислены в справочном разделе в Приложении А.

Полный список числовых идентификаторов блоков — на Вики-странице Minecraft: http://minecraft.gamepedia.com/Blocks[1].

Дополнительные значения, которые можно передавать при создании блоков: http://minecraft.gamepedia.com/Data_values.

Блок типа «шерсть» (`WOOL`) очень полезен для строительства, как вы уже убедились, когда строили дома с коврами случайного цвета. Ниже перечислены все цвета, которые можно использовать с блоками типа `block.WOOL.id`. Вернитесь назад, к программе *buildStreet2.py*, и вспомните, как передаются дополнительные данные в Minecraft API.

0 белый	1 оранжевый	2 малиновый	3 светло-синий
4 желтый	5 светло-зеленый	6 розовый	7 серый
8 светло-серый	9 бирюзовый	10 фиолетовый	11 синий
12 коричневый	13 зеленый	14 красный	15 черный

Дополнительные приключения со строительством

В этом Приключении вы узнали, как создать один блок и группу блоков в мире Minecraft с помощью одной строки в программе на Python, и уже построили несколько внушительных сооружений. Вы познакомились с функциями, которые позволяют разбить программу на небольшие логические фрагменты, а также с циклами `for`, повторяющими свои инструкции. Обладая этими знаниями, вы можете построить почти все что угодно!

- Использовав приемы, изученные в этом приключении, напишите по одной функции для создания каждой из шести граней кубика для игры. Напишите цикл, выполняющий случайное число итераций, каждый раз показывающий разные грани кубика. Используйте `random.randint()` для выбора случайного числа итераций; попробуйте сами и предложите друзьям угадать, какая грань будет показана в следующий раз.
- Напишите список, что еще можно сделать случайным на улице, заполненной домами. Возможно, вам придется присмотреться к зданиям на реальной улице, чтобы понять, чем они отличаются друг от друга. Определите дополнительные функции строительства домов — по одной для каждого типа и попробуйте построить более сложную улицу с домами, отличающимися друг от друга архитектурой.
- Соберитесь с друзьями и объедините разные программы строительства домов в одну большую программу. Попробуйте с ее помощью построить множество домов разного типа в своем мире. Представьте себе это чудо: вы ходите по новому городу, где много домов с разной архитектурой!

[1] http://minecraft-book.ru/id/. — *Примеч. пер.*

Новое достижение: архитектор необыкновенных зданий и строитель потрясающих сооружений в мире Minecraft.

В СЛЕДУЮЩЕМ ПРИКЛЮЧЕНИИ

В Приключении 4 вы узнаете, как взаимодействовать с блоками, в том числе как определить тип блока, на котором стоит персонаж, и узнать, какой блок выбрал игрок. Эти новые знания вы будете использовать для создания захватывающей игры с поисками сокровищ!

Приключение 4
Взаимодействие с блоками

ОДИН ИЗ СПОСОБОВ сделать игру Minecraft интереснее — заставить ее менять свое поведение в зависимости от ситуации вокруг персонажа. Так как персонаж перемещается в игровом мире, вы будете постоянно сталкиваться с необходимостью делать выбор, который связан с ранее принятыми решениями. Поэтому игра каждый раз будет протекать иначе. Прикладной интерфейс Minecraft API позволяет взаимодействовать с блоками: определять тип блока, на котором стоит персонаж, и момент, когда вы касаетесь блока своим мечом.

В этом Приключении вы сначала познакомитесь с основами взаимодействия, написав программу волшебного моста. Его особенность в том, что он позволяет ходить по воде, передвигаться по воздуху и возникает перед персонажем, не давая ему утонуть или упасть. Вскоре ваш мир Minecraft заполнится мостами! Вторая версия программы-моста использует списки языка Python для запоминания места, где построен мост, и заставляет его исчезать на ваших глазах, когда персонаж оказывается на земле и в безопасности.

Наконец, вы узнаете, как определить, выбран ли блок, а потом создадите захватывающую игру поиска сокровищ с подсказками и подсчетом очков, где используете свой волшебный мост для сбора сокровищ, появляющихся в случайных местах и в воздухе.

В этом приключении вы будете работать как заправский инженер-программист: сначала построите все необходимые функции, а в конце объедините их в большую программу. Итак, пристегните ремни и внимательно следите за инструкциями. Предстоит головокружительный полет!

Выяснение информации о блоке, на котором стоит персонаж

В Приключении 2 вы узнали, что местонахождение персонажа в Minecraft можно определить с помощью функции `getTilePos()` и описать координатами *x*, *y* и *z*. По координатам Стива вы уже определяли, встал ли он на волшебный коврик, находится ли на поле, для которого определена геозона.

Однако если программа не хранит подробную карту с расположением всех блоков, одного местоположения персонажа будет недостаточно с учетом роста сложности программ. Вам ничто не мешает хранить собственную карту мира. Но зачем такие сложности? Разве Minecraft не аккумулирует в памяти компьютера необходимую информацию, подготовленную для отображения трехмерного мира на экране?

К счастью, Minecraft API включает в себя функцию `getBlock()`, дающую полный доступ к карте мира Minecraft в памяти. Она готова сообщить вам все о любом блоке — не только о том, на котором стоит Стив, но и о любом в мире Minecraft.

В Приключении 3 вы узнали, что, управляя типом блока, в мире Minecraft можно менять любые блоки. К счастью, `setBlock()` и `getBlock()` действуют как пара: если использовать функцию `setBlock()`, передав ей некоторый числовой идентификатор типа блока, и тут же использовать `getBlock()` с координатами этого же блока, она вернет тот же самый идентификатор.

Скоро вы создадите еще одну захватывающую игру внутри Minecraft, но эта программа будет достаточно большой. Лучший способ формирования больших программ — создать много маленьких программ, а затем объединить их. Давайте претворим этот подход в жизнь и начнем с простой программы, которая определяет, безопасно ли встать на тот или иной блок.

Выясняем, насколько твердая почва под ногами

Запустите Minecraft, IDLE и сервер. У вас должно быть достаточно навыков, чтобы выполнить эти операции самостоятельно, но если понадобится освежить их в памяти, загляните в Приключение 1. Сейчас вы займетесь созданием программы, которая позволит получить важную информацию о месте, где находится персонаж. Прежде чем начать строить мост в воздухе, надо узнать, есть ли твердый грунт под ногами персонажа.

1. Создайте новое окно, выбрав в меню пункт File ▸ New File (Файл ▸ Создать файл). Сохраните новую программу с именем *safeFeet.py*. Не забывайте, что программы нужно сохранять в папке *MyAdventures*, иначе они не будут работать.
2. Импортируйте необходимые модули. Вам потребуется обычный модуль `minecraft` и, так как вы собираетесь работать с блоками, модуль `block`. Вам также требуется небольшая задержка по времени, чтобы замедлить работу, поэтому импортируйте модуль `time`:

```
import mcpi.minecraft as minecraft
import mcpi.block as block
import time
```

3. Подключитесь к игре Minecraft:

```
mc = minecraft.Minecraft.create()
```

4. Части этой программы планируется использовать для создания большой программы, поэтому основная часть ее кода помещается в функцию с именем `safeFeet()`. Это упростит повторное использование данного кода в будущем. Первое, что должна сделать функция, — получить координаты Стива:

```
def safeFeet():
  pos = mc.player.getTilePos()
```

5. `getBlock()` вернет тип блока с указанными вами координатами. Так как `pos.x`, `pos.y` и `pos.z` — координаты персонажа, функции следует передать `pos.y-1`, чтобы получить координаты блока под его ногами:

```
b = mc.getBlock(pos.x, pos.y-1, pos.z pos.z) # обратите внимание: pos.y-1 ↩
 это важно!
```

6. Установим простое правило, которое позволит сказать, находится ли персонаж в безопасности: если под ним воздух или вода, это небезопасно. Во всех остальных случаях можно считать, что персонаж в безопасности. Проверить условие небезопасности гораздо проще, чем сравнивать тип блока с несколькими сотнями безопасных типов, чтобы убедиться в обратном. Строка получилась довольно длинной и ей не хватило ширины книжной страницы. Однако вы должны ввести весь следующий код в одной строке:

```
if b == block.AIR.id or b == block.WATER_STATIONARY.id ↩
 or b == block.WATER_FLOWING.id:
```

7. Если блок имеет один из небезопасных типов, выведите в чат Minecraft предупреждение: персонаж в опасности.

```
    mc.postToChat("not safe")
else:
    mc.postToChat("safe")
```

Будьте внимательны при оформлении отступов: если допустите ошибку, программа не будет работать. Не забывайте, что весь код внутри функции должен иметь один отступ, а любой код внутри инструкции `if` или `else` — еще один, дополнительный, отступ. Инструкции `if` и `else` должны иметь одинаковое количество отступов, так как они взаимосвязаны и находятся на одном логическом уровне.

8. На этом функция `safeFeet()` заканчивается. Оставьте пустую строку в конце функции как напоминание для себя самого, что функция закончилась, и начните игровой цикл `while` без отступа. Как и в предыдущих программах, добавьте короткую задержку в начало цикла и затем используйте новую функцию `safeFeet()`, которая выполнит остальную работу.

```
while True:
  time.sleep(0.5)
  safeFeet()
```

Теперь выберите пункт меню File ▸ Save (Файл ▸ Сохранить), чтобы сохранить программу, и запустите ее, выбрав в меню редактора пункт Run ▸ Run Module (Запустить ▸ Запустить модуль).

Что будет происходить во время движения персонажа по миру Minecraft? Вы должны увидеть, как в чате Minecraft появится слово «safe» (безопасно) или «not safe» (небезопас-

но) в соответствующих местах (рис. 4.1). Попробуйте полетать в воздухе или поплавать в море и посмотрите, что произойдет.

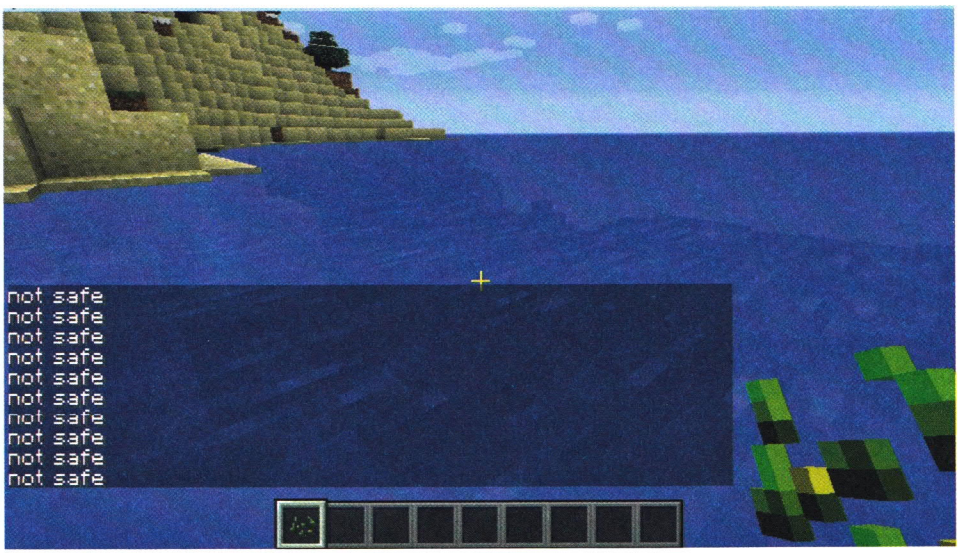


Рис. 4.1. Программа safeFeet говорит о том, что персонаж не в безопасности, когда находится в воздухе

В программе *safeFeet.py*, как в любой другой, в главном игровом цикле используется короткая задержка. Это требуется не всегда, и далее вы встретитесь с программами, которые из-за такой задержки работают неправильно. Однако если программа выводит сообщения в чат, такая задержка, безусловно, полезна, так как позволяет немного притормозить события; иначе чат заполнится сообщениями очень быстро. Попробуйте уменьшить задержку или убрать ее совсем и посмотрите, что из этого получится.

Строительство волшебных мостов

В предыдущей программе вы написали функцию, которая определяет безопасность блока под персонажем и выводит сообщение в чат. Это был маленький шаг на пути к большой программе. Теперь давайте посмотрим, можно ли превратить данный эксперимент во что-то по-настоящему полезное, объединив его с приемами использования функции `setBlock()`, описанными в Приключении 3.

Внеся небольшие изменения в программу *safeFeet.py*, можно заставить ее строить волшебный стеклянный мост под ногами персонажа, чтобы не дать ему утонуть в море или упасть

с высоты. Вы будете использовать новую функцию в следующих программах и в данном Приключении, поэтому внимательно проверьте, правильно ли записали ее имя.

1. Выберите пункт меню File ▸ Save As (Файл ▸ Сохранить как) и сохраните программу *safeFeet.py* с именем *magicBridge.py*.
2. Измените имя функции `safeFeet()`, чтобы она называлась `buildBridge()`, и измените инструкцию `if/else`, как показано ниже жирно, удалив инструкцию `mc.postToChat()` и заменив ее инструкцией `mc.setBlock()`. Теперь всякий раз, когда персонаж ступит на небезопасный блок, под ним появится стеклянный мост. Будьте внимательны при наборе длинной строки в инструкции `if`:

```
def buildBridge():
  pos = mc.player.getTilePos()
  b = mc.getBlock(pos.x, pos.y-1, pos.z)
  if b == block.AIR.id or b == block.WATER_FLOWING.id ↩
    or b==block.WATER_STATIONARY.id:
    mc.setBlock(pos.x, pos.y-1, pos.z, block.GLASS.id)
```

   Здесь удалены инструкции `else` и `mc.postToChat()`, так как они больше не нужны.
3. Если мост будет появляться с большой задержкой, ваш персонаж упадет или утонет, поэтому нужно убрать задержку в начале игрового цикла и, кроме того, задействовать новую функцию `buildBridge()`:

```
while True:
  buildBridge()
```

4. Сохраните программу, выбрав в меню редактора пункт File ▸ Save (Файл ▸ Сохранить).

Запустите программу и побродите по миру, попрыгайте вверх и войдите в воду. Как только под ногами персонажа окажутся воздух или вода, немедленно появится стеклянный мост, который не даст персонажу упасть (рис. 4.2). Теперь он может ходить по воде. Разве это не чудо!

Вопреки вашим возможным ожиданиям, эта программа реагирует на перемещения персонажа не слишком быстро, и он по-прежнему может упасть или утонуть — из-за того, что стеклянный блок появится под ногами с опозданием. Поэтому не торопитесь пускать своего персонажа бегом; не надейтесь, что волшебный мост обезопасит его в любой момент! Экспериментируйте, увеличивая и уменьшая скорость движения, попробуйте режим замедленного перемещения (нажмите и удерживайте клавишу Shift на клавиатуре во время перемещений).

В программе *magicBridge.py* вы убрали задержку из главного цикла. Что произойдет, если вернуть ее на место? Поэкспериментируйте с разными значениями задержки в диапазоне от `0,1` до `2` секунд и посмотрите, как это повлияет на удобство игры. Какое значение обеспечивает максимальное удобство?

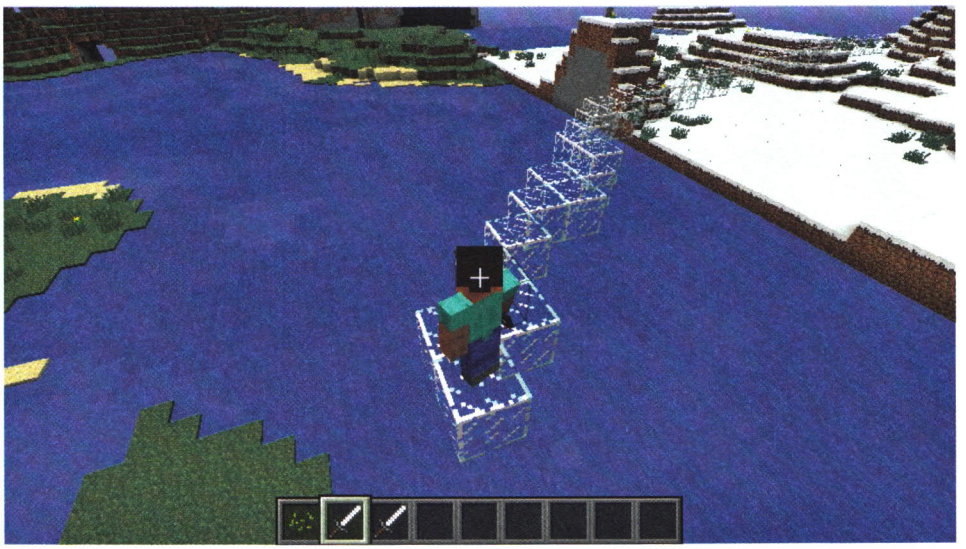


Рис. 4.2. Программа magicBridge.py строит стеклянный мост, что позволяет персонажу ходить по воде

УГЛУБЛЯЕМСЯ В КОД

Остановимся на минуту, так как нужно еще кое-что пояснить относительно функций.

С самого начала книги вы используете функции, такие как `mc.postToChat()` и `mc.getTilePos()`, являющиеся частью Minecraft API. Также с помощью инструкции `def` вы определили собственные функции такие как функция `house()`, созданная в Приключении 2. Что происходит, когда вы используете в своих программах функции?

Мы слегка погрешили против истины, сказав «использовать функцию». Всякий раз, помещая функцию, такую как `house()` или `buildBridge()`, в программный код, вы можете говорить «**вызвать** функцию». Что же в действительности происходит?

Во время выполнения программы на языке Python компьютер запоминает, какая инструкция выполняется в текущий момент, как бы указывая невидимым пальцем на выполняемую строку. Обычно этот «палец» перемещается по тексту программы сверху вниз. Когда в программе используется цикл, он перепрыгивает обратно, в начало цикла, и снова двигается вниз по строкам программы.

Когда вызывается функция, происходит кое-что необычное. Python запоминает, где находился невидимый палец (представьте, что он ставит в этом месте цветной флажок), и перепрыгивает в вызванную функцию, например такую, как `buildBridge()`. Дойдя до конца функции `buildBridge()`, он перепрыгнет обратно к цветному флажку и продолжит выполнять программу.

Использование функций в программах дает массу удобств, два из которых наиболее существенны:

- вы можете разбить программу на мини-программы;
- вы можете многократно использовать один и тот же программный код, заключенный в функцию, из разных мест в программе.

Оба названных обстоятельства упрощают чтение и изменение программ.

Когда **вызывается** функция, Python запоминает, откуда она вызвана, и на время перепрыгивает в точку, где находится определение `def` функции. Достигнув ее конца, Python перепрыгивает обратно, к инструкции, следующей непосредственно за вызовом функции.

Использование списков Python в качестве волшебной памяти

До сих пор во всех программах для хранения информации, меняющейся в ходе выполнения, использовались переменные. Каждая переменная имела свое имя и значение, а если программе требовалось запомнить несколько значений, в ней использовалось несколько переменных.

Однако эти переменные имели фиксированный размер; каждая могла хранить всего одно значение (например, число или строку с текстом). Во многих программах, которые вы будете писать, понадобится хранить неопределенное число элементов: количество необходимых элементов не всегда известно заранее.

К счастью, как любой другой современный язык программирования, Python обладает инструментом, который называется **список**. Список может хранить произвольный объем данных во время выполнения программы.

Список — тип переменной, который может хранить любое количество элементов данных. Вы можете добавлять в список новые элементы, определять длину списка, обращаться к элементам на определенных позициях и удалять их из списка, выполнять множество других операций. Списки — очень удобный способ хранения коллекций одинаковых элементов в одном месте, таких как списки чисел, списки членов групп пользователей и даже списки блоков в игре Minecraft.

Эксперименты со списками

Лучший способ понять суть списков — поэкспериментировать с ними в интерактивной оболочке Python.

В этом разделе особое внимание уделяйте скобкам. Здесь используются два вида скобок: круглые ( ) и квадратные [ ]. Не пугайтесь заранее: разница между этими видами скобок к концу раздела станет для вас очевидной.

1. Переведите окно интерактивной оболочки Python на передний план, щелкнув на нем. Щелкните мышью правее последней строки приглашения к вводу >>>, куда вы начнете вводить интерактивные команды. Не забудьте перед этим остановить выполняющуюся программу, если она была запущена, выбрав в меню пункт Shell ▸ Restart Shell (Оболочка ▸ Перезапустить оболочку) или нажав клавиши CTRL и C. Если этого не сделать, следующие шаги не начнут работать!
2. Создайте новый пустой список и покажите, что это список:

```
a = [] # пустой список
print(a)
```

3. Добавьте в список новый элемент и выведите его еще раз:

```
a.append("hello")
print(a)
```

4. Добавьте второй элемент и снова выведите список:

```
a.append("minecraft")
print(a)
```

5. Узнайте длину списка:

```
print(len(a))
```

6. Получите элементы из списка с определенными позициями. Эти позиции называют **индексами**:

```
print(a[0]) # [0] — индекс первого элемента
print(a[1]) # [1] — индекс второго элемента
```

7. Удалите слово в конце списка и покажите, какое слово осталось в списке:

```
word = a.pop()
print(word)
print(a)
```

8. Узнайте длину списка, а также длину слова:

```
print(len(a))
print(len(word))
```

9. Удалите из списка последний элемент, покажите его и узнайте длину списка:

```
word = a.pop()
print(word)
print(a)
print(len(a))
```

На рис. 4.3 показан вывод, полученный в ходе экспериментов в интерактивной оболочке Python.

Индекс — число, определяющее номер элемента в списке, к которому обращается программа. Индексы указываются в квадратных скобках: например, `a[0]`, чтобы получить первый элемент, и `a[1]`, чтобы получить второй. Нумерация индексов в Python начинается с `0`, то есть индекс `0` всегда соответствует первому элементу в списке. В только что выполненных шагах вы провели операцию *индексирование* списка.

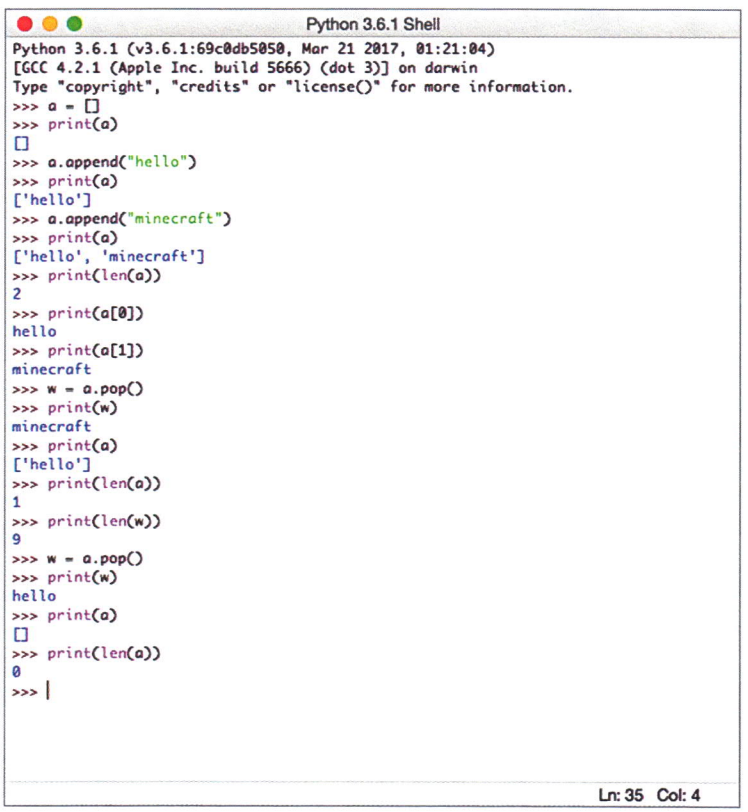


Рис. 4.3. Эксперименты со списками в интерактивной оболочке Python

При добавлении нового элемента в список и в случае удаления существующего длина списка меняется. Что произойдет, если обратиться к несуществующему элементу? Попробуйте провести такой эксперимент в интерактивной оболочке Python и посмотрите, что будет:

```
b = []
print(b[26])
```

Как вы думаете, можно воспрепятствовать произошедшему в программах? (Имейте в виду, что на этот вопрос может быть несколько правильных ответов; поищите разные варианты предотвращения проблемы в интернете.)

Списки в языке Python — это универсальный тип переменных, потому что в них можно хранить элементы разных типов. Однако это не единственный тип переменных в языке Python, способный на хранение. На данном этапе главное, что вы должны запомнить, — не надо заранее знать, насколько большой получится список. Списки растут и сжимаются автоматически, при добавлении элементов в конец функцией `append()` и удалении с конца функцией `pop()`.

Функция **pop** удаляет из списка последний элемент.

Строительство исчезающих мостов с помощью списка Python

Теперь вам предстоит воспользоваться недавно обретенными знаниями, чтобы написать программу строительства мостов, которые будут исчезать, как только персонаж окажется на твердой земле. Эта программа напоминает программу `magicBridge.py`, поэтому вы можете сохранить ее с новым именем и исправить. Мы, в свою очередь, покажем программу целиком, чтобы потом было проще объяснять разные шаги. Используйте операции копирования и вставки в программе *magicBridge.py*, если хотите сэкономить время на вводе.

На сайте книги есть видеоурок, где показано, как написать программу и играть в игру, где строится волшебный мост. Для этого посетите веб-сайт www.wiley.com/go/adventuresinminecraft2e и выберите видеоурок для Приключения 4.

1. Создайте новый файл, выбрав в меню пункт File ▸ New File (Файл ▸ Создать файл), и сохраните его с именем *vanishingBridge.py*, выбрав в меню редактора пункт File ▸ Save As (Файл ▸ Сохранить как).

2. Импортируйте необходимые модули:

```
import mcpi.minecraft as minecraft
import mcpi.block as block
import time
```

3. Подключитесь к игре Minecraft:

```
mc = minecraft.Minecraft.create()
```

4. Создайте пустой список `bridge`. На данный момент мост (bridge) еще не построен, поэтому список блоков моста должен быть пуст:

```
bridge = []
```

5. Определите функцию `buildBridge()` строительства моста. Эта функция будет использоваться в заключительной программе этого Приключения, поэтому постарайтесь ввести имя функции без ошибок. По большей части функция `buildBridge()` повторяет аналогичную функцию в программе *magicBridge.py*, поэтому вы можете просто скопировать ее, чтобы сэкономить время на вводе с клавиатуры. Будьте внимательны при оформлении отступов:

```
def buildBridge():
  pos = mc.player.getTilePos()
  b = mc.getBlock(pos.x, pos.y-1, pos.z)
  if b == block.AIR.id or b == block.WATER_FLOWING.id ↩
    or b == block.WATER_STATIONARY.id:
    mc.setBlock(pos.x, pos.y-1, pos.z, block.GLASS.id)
```

6. После создания очередного блока моста нужно запомнить координаты этого блока, чтобы его можно было удалить, когда персонаж ступит на твердую землю. Для хранения трех координат блока используется еще один список, `coordinate`, который добавляется в список `bridge`. Полное описание происходящего здесь вы найдете во врезке «Углубляемся в код»:

```
    coordinate = [pos.x, pos.y-1, pos.z]
    bridge.append(coordinate)
```

7. Чтобы удалить мост после того, как персонаж сойдет с него, нужна инструкция `else`, которая будет выполняться, как только персонаж окажется на земле. Когда это произойдет, программа начнет удалять блоки моста. Прежде всего, она проверит наличие блоков в мосте, иначе, если попытаться удалить блок из пустого списка, будет сгенерирована ошибка. Инструкция `elif` — сокращенная форма записи «else if», а оператор `!=` означает «не равно». Будьте внимательны при оформлении отступов: инструкция `elif` должна иметь один отступ как часть функции `buildBridge()`; следующая за ней инструкция `if` должна иметь два отступа, так как является частью инструкции `elif`:

```
  elif b != block.GLASS.id:
    if len(bridge) > 0:
```

8. Следующие несколько строк должны иметь три отступа, потому что являются частью инструкции `if`, которая сама является частью инструкции `elif`, являющейся частью функции `buildBridge()`! Уф!

   Помните, как выше добавляли три координаты блока в список `coordinate`? Теперь вам нужно индексировать этот список: `coordinate[0]`, чтобы получить координату *x*; `coordinate[1]`, чтобы получить координату *y*, и `coordinate[2]`, чтобы получить координату *z*. Дополнительная инструкция `time.sleep()` замедлит процесс удаления моста, чтобы его можно было наблюдать:

```
coordinate = bridge.pop()
mc.setBlock(coordinate[0], coordinate[1],
coordinate[2], block.AIR.id)
time.sleep(0.25)
```

9. Наконец, напишите главный игровой цикл. Как и в предыдущих экспериментах, попробуйте использовать разные задержки в игровом цикле, чтобы игра достаточно быстро реагировала на перемещения персонажа. Не забудьте, что игровой цикл является частью основной программы (инструкция `while True:` не должна иметь отступов), поэтому будьте внимательны при оформлении отступов:

```
while True:
  time.sleep(0.25)
  buildBridge()
```

   Сохраните программу, выбрав в меню редактора пункт File ▸ Save (Файл ▸ Сохранить), и затем запустите ее, выбрав в меню пункт Run ▸ Run Module (Запустить ▸ Запустить модуль). Походите по миру Minecraft, направьте персонажа по воде или в обрыв, затем верните на твердую почву. Что произошло? На рис. 4.4 показано, как мост начал исчезать после того, как Стив ступил на землю.

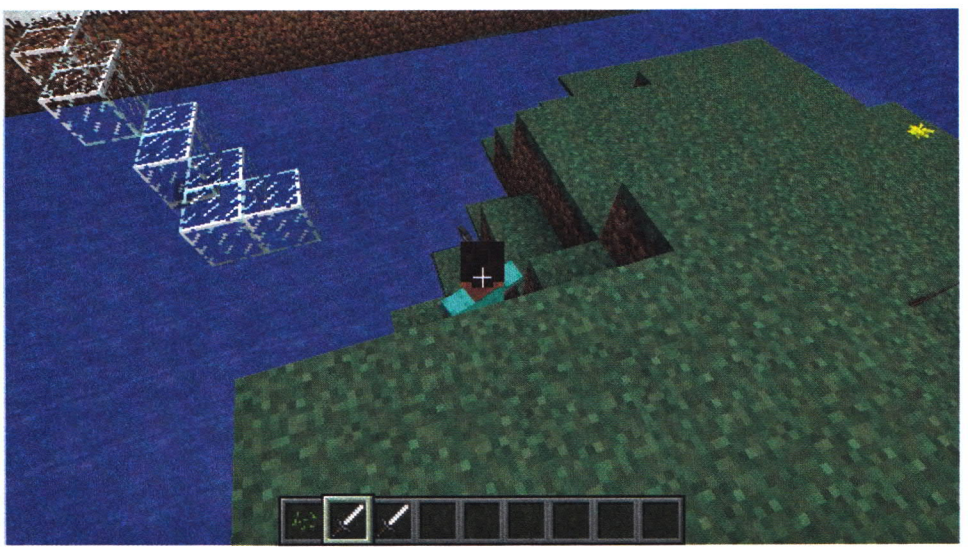


Рис. 4.4. Мост исчезает как по волшебству, как только персонаж оказывается в безопасности

УГЛУБЛЯЕМСЯ В КОД

Экспериментируя со списками в интерактивной оболочке Python, вы добавляли в список текстовые строки. В программе `vanishingBridge.py` используется чуть больше магии языка Python, что требует дополнительных пояснений.

Списки в языке Python могут хранить данные любых типов: текстовые строки, числа и даже другие списки. Вы уже использовали эту возможность в своих программах, вероятно, даже не заметив этого.

Следующая строка создаст список с тремя элементами (координатами *x*, *y* и *z* только что созданного блока):

```
coordinate = [pos.x, pos.y-1, pos.z]
```

Заключая значения в квадратные скобки `[ ]`, вы создаете не пустой список, а список с этими значениями в нем. То же самое можно сделать иначе:

```
coordinate = []
coordinate.append(pos.x)
coordinate.append(pos.y-1)
coordinate.append(pos.z)
```

Потом, когда программе потребуется получить координаты блока для удаления, она извлечет из списка `bridge` список `coordinate`:

```
coordinate = bridge.pop()
```

То есть список хранит три координаты:

`coordinate[0]` — хранит координату *x*-блока; `coordinate[1]` — координату *y*-блока и `coordinate[2]` — координату *z*-блока.

На рис. 4.5 показано, как выглядит этот «список списков».

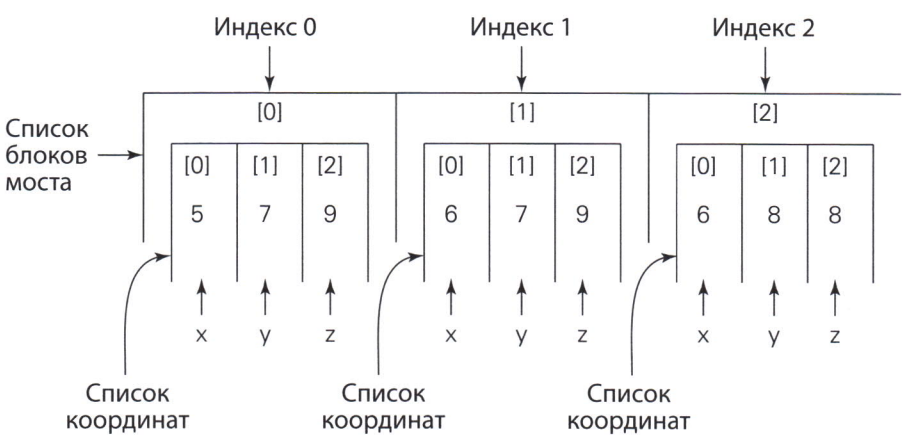


Рис. 4.5. Мост — это список блоков, а каждый блок — список координат

В языке Python есть несколько разных типов переменных для хранения коллекций элементов — помимо списков, которые вы только что использовали. Для хранения координат в этой программе профессиональные программисты на Python воспользовались бы **кортежами**. Мне не хочется вводить сразу много новых понятий, Мартин познакомит вас с кортежами в Приключении 8. Желающие могут поискать информацию о кортежах в интернете и самостоятельно выяснить, чем они отличаются от списков и какие дополнительные преимущества могут привнести в программу на Python.

Определение выбора блока

Последняя особенность, связанная с блоками, которая вам потребуется в этом Приключении, — возможность определять факт выбора блока игроком. Это умение позволит создавать по-настоящему захватывающие игры и программы, даст игроку шанс прямо взаимодействовать с любыми блоками в мире Minecraft.

Создайте новую программу для этого Приключения, чтобы провести эксперимент, после которого вы подойдете к созданию заключительной игры в этом Приключении.

1. Создайте новый файл, выбрав в меню редактора пункт File ▸ New File (Файл ▸ Создать файл). Выберите в меню пункт File ▸ Save As (Файл ▸ Сохранить как) и сохраните программу с именем *blockHit.py*.
2. Импортируйте необходимые модули:

```
import mcpi.minecraft as minecraft
import mcpi.block as block
import time
```

3. Подключитесь к игре Minecraft:

```
mc = minecraft.Minecraft.create()
```

4. Определите позицию персонажа и сместите координаты на один блок в сторону. Используйте новую позицию со смещением для создания алмаза — это будет ваше волшебное сокровище:

```
diamond_pos = mc.player.getTilePos()
diamond_pos.x = diamond_pos.x + 1
mc.setBlock(diamond_pos.x, diamond_pos.y, diamond_pos.z,
            block.DIAMOND_BLOCK.id)
```

5. Определите функцию с именем `checkHit()`. Она будет использоваться в заключительной программе, поэтому постарайтесь ввести имя без ошибок:

```
def checkHit():
```

6. Запросите у Minecraft API список произошедших событий. Это самый обычный список Python, подобный тем, которые вы уже использовали в программе *vanishingBridge.py*:

```
  events = mc.events.pollBlockHits()
```

7. Прочитайте каждое событие по очереди с помощью цикла `for`. Подробно эта форма цикла `for` будет описана во врезке «Углубляемся в код»:

```
for e in events:
  pos = e.pos
```

8. Проверьте, совпадает ли позиция блока, выбранного игроком с помощью меча, с позицией алмаза. Если совпадает, выведите сообщение в чат Minecraft:

```
if pos.x == diamond_pos.x and pos.y == diamond_pos.y ↩
  and pos.z == diamond_pos.z:
  mc.postToChat("HIT")
```

9. Наконец, напишите игровой цикл. На этот раз добавьте в него секундную задержку, чтобы чат Minecraft не заполнился сообщениями слишком быстро. Впрочем, вы можете поэкспериментировать с другими задержками и подобрать более удобную для игры. Будьте внимательны при оформлении отступов:

```
while True:
  time.sleep(1)
  checkHit()
```

Сохраните программу, выбрав в меню редактора пункт File ▸ Save (Файл ▸ Сохранить), и затем запустите ее, выбрав в меню пункт Run ▸ Run Module (Запустить ▸ Запустить модуль).

Покрутитесь на месте, пока не увидите алмазный блок. Потом коснитесь каждой его грани своим мечом. Что произошло? Как показано на рис. 4.6, когда персонаж касается мечом алмаза, в чате Minecraft появляется сообщение «HIT» («Выбран»).

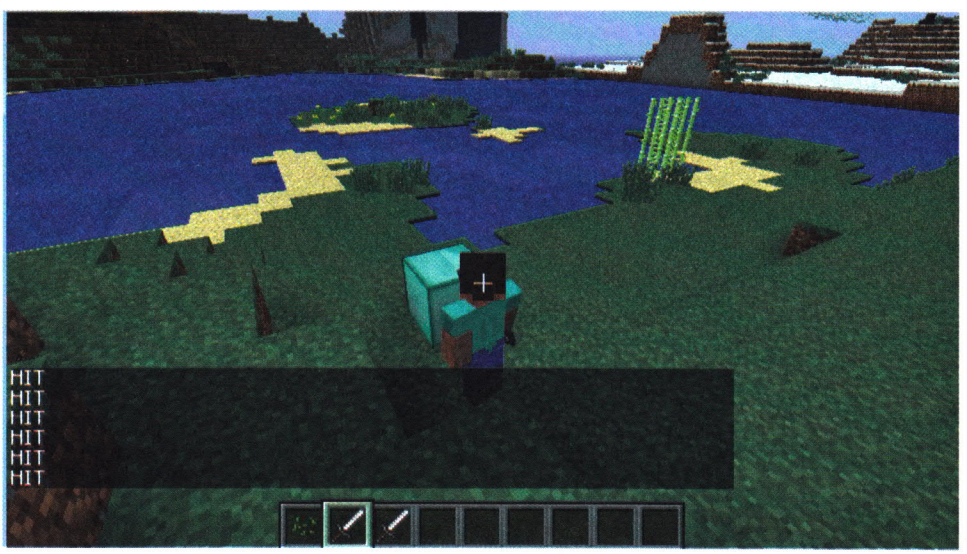


Рис. 4.6. Блок выбран

Если по неосторожности вы нажали не ту кнопку мыши и удалили сокровище, вы не сможете его выбрать! В этом случае перезапустите программу или вручную создайте новый блок сокровища в том же месте, перед вами. Эта необходимость объясняется тем, что для блоков типа `AIR` не генерируется событие выбора блока.

Измените программу *blockHit.py* так, чтобы в цикле `for` она тоже читала значение из переменной `e.face`. В переменной `e.face` Minecraft API возвращает номер одной из шести граней блока, которой коснулся персонаж. Добавьте вывод номера грани из `e.face` в чат Minecraft и затем попробуйте коснуться каждой грани, чтобы выяснить, какие номера им присвоены. Наконец, измените программу так, чтобы она выводила разные сообщения для разных граней алмазного блока.

УГЛУБЛЯЕМСЯ В КОД

В программе определения выбора блока используется цикл `for`, но это другая разновидность цикла, которую вы еще не использовали. Рассмотрим ее подробнее, так как она является довольно мощной особенностью Python.

Обычный цикл `for`, использовавшийся вами прежде, выглядит так:

```
for i in range(6):
  print(i)
```

`range(6)` — это функция, генерирующая список чисел `[0,1,2,3,4,5]`.

Инструкция `for/in` в Python выполняет цикл по всем элементам в списке: сохраняет первый элемент в управляющей переменной цикла (в предыдущем примере это переменная `i`) и выполняет тело цикла, затем сохраняет следующий элемент в управляющей переменной цикла и вновь выполняет тело цикла. Так продолжается, пока список не будет полностью исчерпан.

Это значит, что если имеется непустой список, вы можете точно таким же способом обойти все его элементы. Выполните следующие инструкции в интерактивной оболочке Python:

```
for name in ["Becky", "Keira", "Phil", "Holly"]:
  print("hello " + name)
```

Точно так же можно выполнить цикл по символам в текстовой строке:

```
name = "Becky"
for ch in name:
  print(ch)
```

Создание игры с поиском сокровищ

Большую часть времени в этом Приключении вы потратили на приобретение новых знаний и создание фрагментов программного кода, чтобы испытать разные возможности, имеющиеся в игре Minecraft. Теперь пришло время «сшить» из полученных фрагментов полноценную игру. Игра, которую вам предстоит написать, называется «Sky Hunt» («Поиск в небе»); в ней игроку с помощью подсказок предлагается найти алмазные блоки, случайным образом разбросанные в воздухе, и собрать их.

У этой игры есть одна хитрость: на каждом шаге персонаж оставляет золотой блок, который будет стоить вам одно очко. Если бесцельно бегать в поисках сокровищ, счет быстро обнулится и даже может стать отрицательным! Вам придется использовать навигационные средства Minecraft, чтобы быстро находить алмазные блоки и добираться до них кратчайшим путем.

Когда вы подберете очередной алмазный блок, к вашему счету добавятся очки, а золотой след, как по волшебству, растает (возможно, на месте золотых блоков останутся отверстия, о которые можно споткнуться, — будьте к этому готовы!).

Эта программа в основном построена из фрагментов экспериментальных программ, которые вы уже написали в данном Приключении. Можете копировать эти фрагменты в новую программу и менять их, чтобы сэкономить время на вводе с клавиатуры. Но мы включили полный текст программы, чтобы вы могли видеть, что необходимо.

Профессиональные программисты часто начинают с создания простого каркаса будущей программы с несколькими инструкциями `print` и проверяют, насколько правильно каркас работает, а затем постепенно добавляют и проверяют новые особенности. В этом разделе вы сами будете действовать как программист, создавая и тестируя программу шаг за шагом. Для начала напишем каркас игрового цикла с несколькими фиктивными функциями, которые вы постепенно замените на настоящие.

Создание функций и главного игрового цикла

1. Создайте новый файл, выбрав в меню пункт File ▸ New File (Файл ▸ Создать файл), и сохраните его с именем *skyHunt.py*, выбрав пункт File ▸ Save As (Файл ▸ Сохранить как).
2. Импортируйте необходимые модули:

```
import mcpi.minecraft as minecraft
import mcpi.block as block
import time
import random
```

3. Подключитесь к игре Minecraft:

```
mc = minecraft.Minecraft.create()
```

4. Создайте переменную `score`, которая хранит количество очков, заработанных в игре. Также создайте константу `RANGE`, которая определит уровень сложности игры, — расстояние от персонажа, на котором будут появляться новые сокровища. На время тести-

рования выберите для константы небольшое значение. Когда программа будет закончена, вы сможете его увеличить:

```
score = 0
RANGE = 5
```

5. Так как разработка и тестирование программы ведутся поэтапно, для начала напишите несколько фиктивных функций — по одной для каждой особенности в программе. Функции в языке Python не могут быть пустыми и должны содержать хотя бы одну инструкцию. В качестве таковых можно использовать инструкции `print`, которые будут выводить сообщения и служить заместителями для кода, который вы напишете потом:

```
treasure_x = None # координата x сокровища

def placeTreasure():
  print("placeTreasure")

def checkHit():
  print("checkHit")

def homingBeacon():
  print("homingBeacon")

bridge = []

def buildBridge():
  print("buildBridge")
```

6. Теперь добавьте главный игровой цикл. В законченной программе он будет работать очень быстро (10 раз в секунду), чтобы получилась аккуратная золотая дорожка, тянущаяся за персонажем. На время тестирования его следует замедлить до одного раза в секунду. Внутри игрового цикла задействуйте фиктивные функции, которые только что написали.

```
while True:
  time.sleep(1)

  if treasure_x == None and len(bridge) == 0:
    placeTreasure()

  checkHit()
  homingBeacon()
  buildBridge()
```

7. Сохраните программу, выбрав в меню пункт File ▸ Save (Файл ▸ Сохранить), и запустите ее, выбрав пункт Run ▸ Run Module (Запустить ▸ Запустить модуль). На данном этапе не должно появиться никаких ошибок, а в окне интерактивной оболочки Python должны начать появляться сообщения с частотой раз в секунду. Теперь у вас есть каркас будущей программы и можно постепенно добавлять в него новый программный код.

Создание сокровищ в небе

Сначала вам нужно написать функцию, создающую алмазный блок в случайном месте в небе. Координаты блока будут храниться в трех глобальных переменных с начальными значениями `None`. Это специальное значение в языке Python, указывающее на то, что переменная присутствует в памяти, но пока ничего не хранит. По этому значению в игровом цикле определяется необходимость создания нового алмазного блока.

1. Создайте глобальные переменные для хранения координат алмазного блока, добавив строки, выделенные жирно:

```
treasure_x = None
treasure_y = None
treasure_z = None
```

2. Добавьте в функцию `placeTreasure()` следующий код (и уберите инструкцию `print`):

```
def placeTreasure():
  global treasure_x, treasure_y, treasure_z
  pos = mc.player.getTilePos()
```

3. Используйте функцию `random.randint`, чтобы выбрать позицию для сокровища не дальше чем в `RANGE` блоках от персонажа. Установите координату *y* так, чтобы она находилась выше персонажа (возможно, блок окажется где-то в небе):

```
  treasure_x = random.randint(pos.x,   pos.x+RANGE)
  treasure_y = random.randint(pos.y+2, pos.y+RANGE)
  treasure_z = random.randint(pos.z,   pos.z+RANGE)
  mc.setBlock(treasure_x, treasure_y, treasure_z, ↩
    block.DIAMOND_BLOCK.id)
```

Запустите программу и проверьте, создала ли она алмазный блок неподалеку от персонажа.

Сбор сокровищ

Теперь добавьте код из программы *blockHit.py* и немного измените его, чтобы он определял факт выбора алмазного блока, когда персонаж коснется его мечом.

1. Удалите инструкцию `print` из функции `checkHit()` и вместо нее вставьте код, приведенный ниже. Переменные `score` и `treasure_x` здесь объявлены как глобальные (`global`), потому что функция `checkHit()` будет менять их значения. Язык Python требует перечислять внутри функций любые глобальные переменные, значения которых они меняют. Если этого не сделать, программа не будет работать:

```
def checkHit():
  global score
  global treasure_x
```

2. Прочитайте список событий выбора блоков и проверьте, совпадают ли координаты какого-либо события с координатами алмазного блока:

```
  events = mc.events.pollBlockHits()
  for e in events:
```

```
    pos = e.pos
    if pos.x == treasure_x and pos.y == treasure_y ↩
      and pos.z == treasure_z:
      mc.postToChat("HIT!")
```

3. Теперь программа должна добавить игроку очки за найденное сокровище и удалить алмазный блок, чтобы он скрылся. Кроме того, не забудьте присвоить переменной `treasure_x` значение `None` (чтобы `placeTreasure()` могла создать новый блок в случайном месте). Будьте внимательны при оформлении отступов, так как этот код является частью инструкции `if`:

```
    score = score + 10
    mc.setBlock(treasure_x, treasure_y, treasure_z,
                block.AIR.id)
    treasure_x = None
```

Сохраните и запустите программу; проверьте, исчезает ли сокровище после того, как персонаж его коснется. Также проверьте, создается ли новый алмазный блок в случайном месте недалеко от персонажа после исчезновения выбранного.

Добавление вывода подсказок

Программа будет раз в секунду выводить в чат Minecraft число очков на счету игрока и примерное расстояние до сокровища. Ниже описано, как добавить его в программу.

1. Создайте переменную `timer`. Так как игровой цикл будет выполняться примерно 10 раз в секунду, нужно отсчитать 10 циклов, чтобы вывести информацию не чаще чем раз в секунду. Переменная `timer` поможет вам в этом. Если скорость работы игрового цикла в будущем изменится, вам придется изменить константу `TIMEOUT`. Следующий код должен быть добавлен выше функции `homingBeacon()` (обратите внимание на отсутствие отступов):

```
TIMEOUT = 10
timer = TIMEOUT
```

2. Удалите инструкцию `print()` из функции `homingBeacon()` и объявите в ней переменную `timer` как глобальную (`global`), потому что эта функция будет менять ее значение:

```
def homingBeacon():
  global timer
```

3. Если где-то в небе уже есть сокровище, переменная `treasure_x` будет хранить значение, отличное от `None`. Вы должны убедиться, что сокровище имеется, иначе в чате Minecraft появится сообщение без расстояния до блока:

```
  if treasure_x != None:
```

4. Эта функция будет вызываться из главного игрового цикла 10 раз в секунду, поэтому выводить подсказку она должна только один раз на каждые 10 вызовов:

```
    timer = timer - 1
    if timer == 0:
        timer = TIMEOUT
```

5. Когда переменная `timer` обнулится (в каждом десятом вызове функции, или один раз в секунду), вычислите примерное расстояние от персонажа до сокровища. Функция `abs()` вернет абсолютное (положительное) значение разности между двумя координатами. Сложив положительные разности координат, вы получите примерное расстояние до сокровища. Будьте внимательны при оформлении отступов, так как следующий код принадлежит самой последней инструкции `if`:

```
pos = mc.player.getTilePos()
diffx = abs(pos.x - treasure_x)
diffy = abs(pos.y - treasure_y)
diffz = abs(pos.z - treasure_z)
diff = diffx + diffy + diffz
mc.postToChat ("score:" + str(score) + ↩
  " treasure:" + str(diff))
```

Сохраните программу, запустите ее и проверьте, выводится ли в чат Minecraft расстояние до сокровища и количество очков. Так как программа находится в процессе разработки и тестирования, игровой цикл работает в 10 раз медленнее. Поэтому на данный момент сообщения должны появляться в чате Minecraft не чаще одного раза в 10 секунд. Проверьте, так ли это, мысленно просчитав от 1 до 10. На данном этапе вы еще будете видеть сообщения раз в секунду в интерактивной оболочке Python — программа не завершена, в ней остаются фиктивные функции.

Добавление строительства моста

Сейчас вам предстоит добавить в программу функции строительства моста из программы *vanishingBridge.py*, написанной ранее. Вам потребуется лишь немного изменить ее, чтобы она проверяла, стоит ли персонаж на золотом блоке, и если нет — чтобы создавала бы такой блок.

1. Исправьте функцию `buildBridge()` так, чтобы она выглядела, как показано ниже. Отличия в коде от этой же функции в программе *vanishingBridge.py* выделены **жирным шрифтом**:

```
bridge = []

def buildBridge():
  global score
  pos = mc.player.getTilePos()
  b = mc.getBlock(pos.x, pos.y-1, pos.z)

  if treasure_x == None:
    if len(bridge) > 0:
      coordinate = bridge.pop()
      mc.setBlock(coordinate[0],
```

```
                coordinate[1],
                coordinate[2],
                block.AIR.id)
        mc.postToChat("bridge:" + str(len(bridge)))
        time.sleep(0.25)
    elif b != block.GOLD_BLOCK.id:
        mc.setBlock(pos.x, pos.y-1, pos.z, block.GOLD_BLOCK.id)
        coordinate = [pos.x, pos.y-1, pos.z]
        bridge.append(coordinate)
        score = score - 1
```

2. Поздравляю! Вы завершили создание программы. Теперь она полностью готова, и задержку `time.sleep(1)` надо изменить в игровом цикле так, чтобы она осуществлялась на 0,1 секунды. Благодаря этому игровой цикл будет работать с частотой 10 раз в секунду. Переменная `timer` в `homingBeacon()` будет вести счет циклов до 10, а сообщения в чате Minecraft начнут появляться чаще — один раз в секунду.

Сохраните и запустите программу. Убедитесь, что золотой след за персонажем исчезает, как только он подбирает сокровище, и с каждым шагом количество очков уменьшается.

Теперь вам осталось только получить удовольствие от игры! Видите, как трудно сохранить хороший счет, собирая сокровища.

На рис. 4.7 показано, как выглядит подсказка с расстоянием и счетом в чате Minecraft.

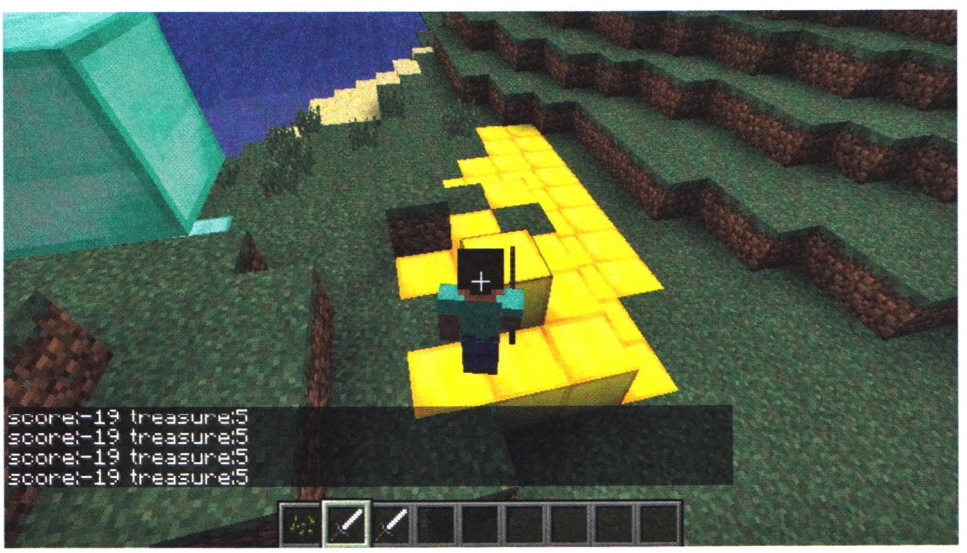


Рис. 4.7. Подсказка, показывающая статистику движения, когда вы играете в Sky Hunt

Краткая справочная таблица

Команда	Описание
`b = mc.getBlock(10, 5, 2)`	Определение типа блока в заданной позиции

`hits = mc.events.pollBlockHits()`
`for hit in hits:`
`  pos = hit.pos`
`  print(pos.x)`
	Определение выбранного блока

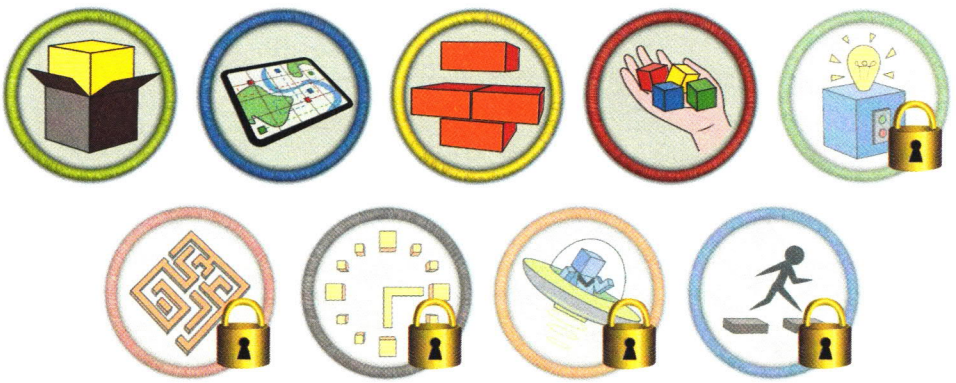
`a = [] # пустой список` `a = [1,2,3] # инициализированный список`	Создание списков
`a.append("hello")`	Добавление в конец списка
`print(a)`	Вывод содержимого списка
`print(len(a))`	Определение размера списка
`print(a[0]) # 0=первый, 1=второй`	Обращение к элементам списка по индексам
`print(a[-1])`	Обращение к последнему элементу списка
`word = a.pop() # удалить последний элемент` `print(word) # только что удаленный элемент`	Удаление последнего элемента из списка
`for item in a:` `  print(item)`	Цикл по всем элементам в списке

Дополнительные приключения с блоками

В этом Приключении вы узнали, как с помощью функции `getBlock()` получить информацию о блоке, на котором стоит персонаж, и как с помощью `events.pollBlockHits()` определить, какого блока он коснулся. Вы создали удивительную и законченную игру внутри Minecraft с подсчетом очков!

- Присвойте константе `RANGE` бо́льшее значение, чтобы алмазные блоки создавались еще дальше от персонажа. Это усложнит игру, поэтому попробуйте добавить новую функцию, которая будет выводить слова «cold» (холодно), «warm» (тепло) или «hot» (горячо) в чат Minecraft — в зависимости от расстояния, отделяющего персонаж от сокровища.

- Придумайте такую схему подсчета очков, чтобы счет получался более разумным. Проверьте на практике, насколько лучше увеличивать счет за каждое найденное сокровище и уменьшать за каждый истраченный золотой блок.
- Познакомьтесь с теоремой Пифагора в интернете и подумайте, можно ли получить более точную оценку расстояния и тем самым сделать функцию `homingBeacon()` более точной. (Скажу по секрету, что Мартин немного рассказывает об этом в Приключении 7, поэтому вы можете заглянуть туда и решить, под силу ли вам это!)



Новое достижение: эксперт по определению законов гравитации и хождению по воде — два чуда в одном Приключении.

В СЛЕДУЮЩЕМ ПРИКЛЮЧЕНИИ

В Приключении 5 вы узнаете, как читать данные из файлов, необходимые для автоматического строительства больших и сложных сооружений, таких как лабиринты Minecraft. Используя новые навыки, вы построите огромный копировальный аппарат для создания трехмерных копий, с помощью которого сможете копировать большие объекты в мире Minecraft. Теперь деревья не будут оказываться внутри стен!

Приключение 5
Использование файлов с данными

ТОЛЬКО КОГДА появляется умение обрабатывать большие объемы данных, программирование становится по-настоящему захватывающим. В этом случае программа превращается в комплекс правил, определяющих, как читать, обрабатывать и отображать на экране данные, которые становятся действительно важным элементом.

В этом Приключении вы сначала научитесь создавать текстовые файлы, описывающие лабиринты. Затем узнаете, как автоматически создавать такие лабиринты в мире Minecraft для себя и друзей, и удивитесь тому, что маленькая программа на Python способна создавать большие и сложные сооружения.

Далее вы разовьете эту идею и создадите виртуальный трехмерный «сканер» и «принтер» — некий копировальный аппарат. Все, что окажется в его камере, можно сохранить в файл, а потом воспроизвести и телепортировать в любую точку мира Minecraft и даже загрузить в Minecraft на другом компьютере! Вы сможете сформировать личную библиотеку объектов и строить их так быстро, что ваши друзья захотят иметь собственные копировальные аппараты.

Чтение данных из файла

Компьютерные программы не отличаются сообразительностью, они лишь следуют инструкциям, которые вы им даете. Если не изменить исходные данные для программы, при запуске она всякий раз будет повторять одно и то же. Программы для Minecraft, которые вы создали в предыдущих приключениях, в значительной степени были диалоговыми: их действия зависели от происходящего в игровом мире, который являлся источником исходных данных.

Что можно делать с файлами данных

Интересный способ заставить компьютерные программы действовать по-разному при каждом запуске — сохранить исходные данные в текстовый файл и организовать его чтение во время запуска программы. Можно создать много текстовых файлов и даже написать меню, чтобы выбирать из него файл для загрузки, в зависимости от желаемого поведения программы. Такие программы называют программами, управляемыми данными, потому что их поведение определяется данными во внешнем текстовом файле.

Если задуматься, можно вспомнить несколько программ, использующих файлы с данными. Текстовый процессор, которым вы пользуетесь для ввода домашних заданий, сохраняет данные в файлах. Когда вы снимаете на цифровую фотокамеру, фотографии сохраняются

в файлах. Графический редактор читает ваши снимки из файла. Даже игра Minecraft использует файлы с данными, например, чтобы сохранить и загрузить мир, а также для хранения текстур всех видов блоков в мире. Использование файлов с данными увеличивает гибкость программы, с их помощью она может пользоваться разными способами; нет необходимости менять ее всякий раз, когда понадобится что-то иное. Кроме того, такими файлами можно делиться с друзьями, если у них есть такая же программа, как у вас.

Первой программой, которую я написал для демонстрации использования файлов с данными в Minecraft, стала программа Minecraft BMP builder, которую можно найти в моем блоге (http://blog.whaleygeek.co.uk/minecraft-pi-with-python/): она читает изображение из файла в формате BMP и воссоздает его блок за блоком в мире Minecraft. С ее помощью я создал огромный логотип Raspberry Pi — настолько большой, что, если смотреть снизу вверх, его верхний край скрывается в тумане! Эта программа может читать любые изображения в формате BMP и воссоздавать их из блоков в мире Minecraft, без внесения каких-либо изменений в программу.

Первым делом вы напишете программу, чтобы научиться открывать и читать текстовые файлы из файловой системы компьютера.

Создание подсказок

Чтобы узнать, как открывать и читать текстовые файлы, напишем программу, выводящую подсказки. Она будет читать подготовленный вами файл с подсказками-советами и выводить их по одной в чат Minecraft через случайные интервалы времени. Вам потребуется текстовый файл с подсказками, поэтому первая задача — создать такой файл. Это можно сделать в редакторе IDLE точно так же, как вы создаете программы на Python.

Запустите Minecraft, IDLE и сервер. У вас должно быть достаточно навыков, чтобы выполнить эти операции самостоятельно, но если понадобится освежить их в памяти, загляните в Приключение 1.

1. В окне IDLE создайте новый текстовый файл, выбрав в меню пункт File ▸ New File (Файл ▸ Создать файл). Сохраните его с именем *tips.txt*.
2. Теперь введите четыре-пять подсказок-советов, каждую в отдельной строке. Ниже даны примеры таких подсказок, но будет гораздо интереснее, если вы напишете свои. Не забывайте нажимать клавишу Enter в конце каждой строки, чтобы каждая подсказка в текстовом файле заканчивалась символом новой строки (больше о символах новой строки вы узнаете далее в этом Приключении):

```
Build yourself a house before the mobs come and get you
Use flowing water to fill underwater channels
Build up with sand then knock out the bottom block
Use both first-person and third-person views
Double tap space bar to fly into the sky
```

Перевод:

```
Постройте себе дом, прежде чем мобы придут и захватят вас
Заполните сухие каналы водой
Создайте конструкцию из песка, затем выбейте нижний блок
Рассматривайте мир глазами Стива и со стороны
Чтобы прыгнуть вверх, нажмите дважды клавишу пробела
```

3. Сохраните этот файл в папке *MyAdventures*. Запускать его не нужно: это не программа на Python, а текстовый файл, который будет использовать ваша программа.

Символ **новой строки** — специальный невидимый символ, который используется компьютерами, чтобы обозначить конец текстовой строки.

Иногда его называют символом *возврата каретки*. Это название появилось в те времена, когда широко использовались механические печатные машинки. Всякий раз, когда требовалось перейти в начало следующей строки, приходилось нажимать клавишу возврата каретки, которая переводила каретку с бумагой в крайнее правое положение (печатная головка оказывалась у левого края листа бумаги) и прокручивала ее на одну строку вниз.

Теперь, имея файл с данными или с исходными данными, можно приступать к созданию программы, которая их обрабатывает. Эта программа будет читать и выводить случайные подсказки через случайные интервалы времени в чат Minecraft. В итоге во время игры в чате периодически будут появляться полезные подсказки.

1. Создайте новый файл, выбрав в меню пункт File ▸ New File (Файл ▸ Создать файл), и сохраните его с именем *tipChat.py*.
2. Импортируйте необходимые модули:

```
import mcpi.minecraft as minecraft
import time
import random
```

3. Подключитесь к игре Minecraft:

```
mc = minecraft.Minecraft.create()
```

4. Определите константу с именем файла, чтобы потом ее было легко изменить для чтения данных из другого файла:

```
FILENAME = "tips.txt"
```

5. Откройте файл в режиме для чтения (подробнее о режимах рассказано ниже, во врезке «Углубляемся в код»):

```
f = open(FILENAME, "r")
```

6. Прочитайте все строки в список с именем `tips`. Напомню, что вы уже использовали списки в Приключении 4 (для строительства волшебного моста):

```
tips = f.readlines()
```

7. Закончив чтение, закройте файл. Старайтесь всегда закрывать файлы, завершив работу с ними: когда файл открыт, он не может использоваться другими программами, например текстовым редактором, которыми вы обычно пользуетесь для ввода текста:

```
f.close()
```

8. Теперь напишите игровой цикл со случайной задержкой от 3 до 7 секунд:

```
while True:
  time.sleep(random.randint(3,7))
```

9. Выберите случайное сообщение из списка `tips` и выведите его в чат. Чтобы убрать из строки нежелательный символ новой строки, воспользуйтесь функцией `strip()`. Детально символ новой строки мы обсудим ниже, а пока не волнуйтесь, если не понимаете, почему вокруг него такая суета.

```
msg = random.choice(tips)
mc.postToChat(msg.strip())
```

Сохраните и запустите программу. Что произошло? Пока вы бродите по миру Minecraft и играете, время от времени в чате будут появляться советы, как показано на рис. 5.1. Обратите внимание на расположение сообщений в чате Minecraft на экране и их регулярное удаление.

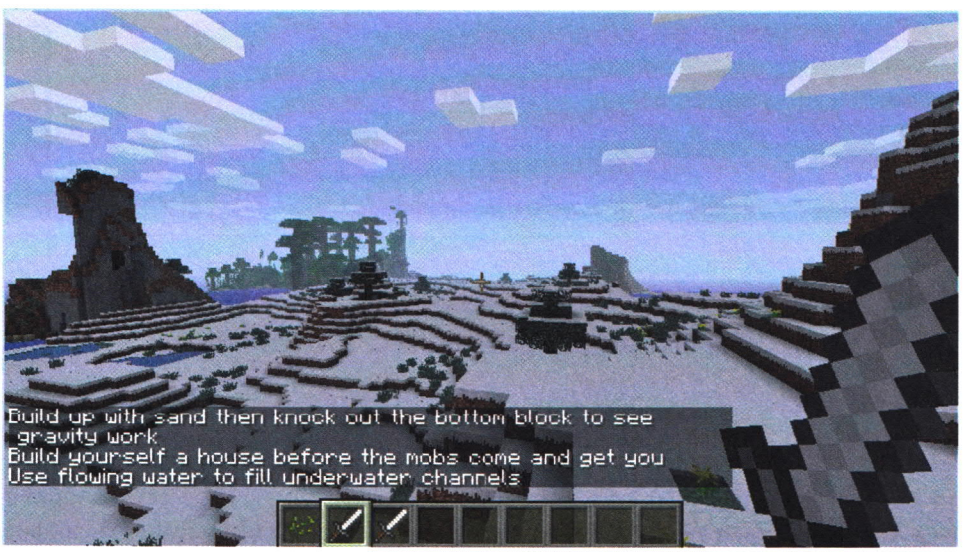


Рис. 5.1. В процессе игры в Minecraft в чате периодически появляются подсказки

Добавьте еще несколько подсказок в файл *tips.txt*, опираясь на собственный опыт игры в Minecraft. Передайте свой файл *tips.txt* и программу *tipChat.py* друзьям, изучающим Minecraft, и научите их пользоваться программой, чтобы они могли быстро получать полезные советы. Вы можете написать целую серию файлов *tips.txt* с подсказками на разные темы и разной степени подробности, в зависимости от подготовленности игрока, и опубликовать их на маленьком веб-сайте вместе с программой *tipChat.py*.

УГЛУБЛЯЕМСЯ В КОД

В программе *tipChat.py* используется немного магии, в строке

```
f = open(FILENAME, "r")
```

функция `open()` открывает файл с именем, хранящимся в константе `FILENAME`, — это понятно. Но что означает `"r"` в конце? Открывая файл, вы должны сообщать функции `open()`, какой уровень доступа к нему необходим. В данном случае аргумент `"r"` указывает, что вам требуется только читать файл. Если по ошибке записать что-то в файл, открытый таким способом, появится сообщение об ошибке. Данный прием защищает файлы от непреднамеренного уничтожения. Если вам нужно открыть файл и что-то записать в него, вместо `"r"` можно передать `"w"` (или, если требуется читать из файла и записывать в него, — `"rw"`).

И еще, что такое `f`? Это простая переменная, которая хранит дескриптор (описание) файла. Дескриптор — своеобразный «рычаг» для управления файлом, виртуальная ссылка на реальный файл в файловой системе компьютера. Если потребуется выполнить операцию с открытым файлом, для ссылки на него используется переменная `f`. Это очень удобно, потому что, как в случае с любыми другими переменными, их можно создать несколько и одновременно открыть, например:

```
f1 = open("config.txt", "r")
f2 = open("levels.txt", "r")
f3 = open("score.txt", "rw")
```

Переменные `f1`, `f2` и `f3` можно использовать в оставшейся части программы для ссылки на файлы, участвующие в операциях чтения или записи.

Создание лабиринтов из файлов с данными

Теперь, когда вы узнали, как в программах читать данные из файлов, можно найти этим знаниям практическое применение. Из опыта предыдущих приключений вы знаете, как просто создать блоки в мире Minecraft. А теперь представьте возможность конструировать блоки на основе списков, хранящихся в файле. Именно этим вы и займетесь: напишите законченную программу строительства трехмерного лабиринта в Minecraft, информация о котором хранится во внешнем файле. Но сначала нужно решить, как эта информация будет представлена в файле.

На сайте книги есть видеоурок, где показано, как написать программу, строящую лабиринт, и как играть с ним. Для этого посетите веб-сайт www.wiley.com/go/adventuresinminecraft2e и выберите видеоурок для Приключения 5.

Как сказано выше, вы научитесь создавать трехмерные лабиринты. Строиться они будут в трехмерном мире Minecraft, хотя в действительности являются двумерными, но построенными из трехмерных блоков. Лабиринты будут иметь только один уровень (этаж). Это значит, что для каждого блока в файле достаточно сохранить координаты *x* и *z*.

Теперь надо решить, как представить в файле блоки. Вам потребуются стены и пустое пространство. Чтобы избежать лишних сложностей, используйте 1 для представления блока стены и 0 — для блока пустого пространства (воздуха). Позднее вы всегда сможете изменить тип блоков для строительства стен в программе на Python.

Файлы CSV

Для представления лабиринтов в виде файлов в данной программе используются текстовые файлы специального типа, которые называют **CSV-файлами**.

CSV-файл содержит значения, разделенные запятыми (Comma-Separated Values). Значения в таких файлах хранятся в простом текстовом виде и отделяются друг от друга запятыми. Можно представить простую таблицу или базу данных в виде CSV-файла. Каждой табличной строке или записи в базе данных соответствует строка в файле, а каждому полю или столбцу — некоторый текст в этой строке, заканчивающийся запятой. В некоторых CSV-файлах первая строка используется для хранения заголовков полей/столбцов. Все популярные электронные таблицы и базы данных поддерживают импортирование и экспортирование данных в формате CSV.

CSV — очень простой и широко используемый формат. Ниже приведен пример CSV-файла, хранящего часть таблицы из базы данных с характеристиками игровых персонажей Minecraft. Первая запись (строка) в этом файле содержит названия полей, остальные строки — данные, разделенные запятыми:

```
Name,Handle,Speciality
David,w_geek,Coding in Python
Roma,physics_gurl,Designing big buildings
Ryan,mr_teck,Minecraft robots
Craig,rrrrrrrr,TNT expert
```

Первая строка в этом файле хранит названия трех полей: `Name`, `Handle` и `Speciality`. Остальные четыре строки — их значения.

При проектировании сложных программных структур иногда полезно использовать вспомогательные инструменты. Для меня как для программиста одним из них является электронная таблица, которой я регулярно пользуюсь для создания и представления данных. Электронные таблицы дают уникальную возможность экспортировать табличные

данные в формате CSV и затем использовать эти файлы в программах. Вы можете попробовать определить лабиринт в своей программе электронной таблицей, установив минимальную ширину столбцов и использовав формулы для отображения белых квадратиков там, где ячейки хранят значение 0, для отображения желтых — значение 1. После создания лабиринта таким визуальным способом экспортируйте его в файл *maze.csv*, запустите программу и попробуйте со своими друзьями найти выход из лабиринта!

В файле с лабиринтом нам не нужны названия полей, потому что все столбцы в таблице представляют собой данные одного типа: в ячейке таблицы хранится 0, если она соответствует пустому пространству, и 1, если она соответствует блоку стены. Файл с примером лабиринта можно загрузить с сайта книги, а для тех, кто все любит делать вручную, его содержимое приведено ниже:

1. Создайте новый текстовый файл, выбрав в меню пункт File ▸ New File (Файл ▸ Создать файл). Выберите пункт меню File ▸ Save As (Файл ▸ Сохранить как) и сохраните файл с именем *maze1.csv*.
2. Аккуратно введите следующие строки — в каждой из них должно быть одинаковое количество столбцов. Если приглядеться к данным, можно даже заметить структуру лабиринта! В файле всего 16 строк, по 16 столбцов в каждой. Вы можете отмечать галочками строки, которые уже введены, чтобы не запутаться:

```
1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1
0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1
1,1,1,1,1,1,1,1,1,0,1,0,1,1,0,1
1,0,0,1,0,0,0,0,1,0,1,0,1,0,0,1
1,1,0,1,0,1,1,0,0,0,0,0,1,0,1,1
1,1,0,1,0,1,1,1,1,1,1,1,1,0,1,1
1,1,0,0,0,1,1,1,1,1,0,0,0,0,1,1
1,1,1,1,1,1,0,0,0,0,0,1,1,1,1,1
1,0,0,0,0,1,0,0,0,0,0,0,0,0,0,1
1,0,1,1,1,1,0,0,0,0,0,1,1,1,1,1
1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1
1,0,1,1,1,1,1,1,1,1,0,1,1,1,1,1
1,0,1,0,0,0,0,0,0,1,0,0,0,0,0,1
1,0,1,0,1,1,1,1,0,1,1,1,1,1,0,1
1,0,0,0,0,0,0,0,0,1,0,0,0,0,0,1
1,1,1,1,1,1,1,1,1,1,0,1,1,1,1,1
```

3. Сохраните файл. Вы будете его использовать, когда напишете соответствующую программу — считывающую и обрабатывающую данные.

Строительство лабиринта

После создания файла с описанием будущего лабиринта можно приступать к написанию программы на Python, которая будет читать данные из этого файла и строить лабиринт в мире Minecraft.

1. Создайте новый файл, выбрав в меню пункт File ▸ New File (Файл ▸ Создать файл), и сохраните его с именем *csvBuild.py*, выбрав в меню пункт File ▸ Save As (Файл ▸ Сохранить как).
2. Импортируйте необходимые модули:

```
import mcpi.minecraft as minecraft
import mcpi.block as block
```

3. Подключитесь к игре Minecraft:

```
mc = minecraft.Minecraft.create()
```

4. Определите несколько констант. Константа GAP — тип блока для пустого пространства. Обычно это AIR (воздух), но вы можете поэкспериментировать с другими типами, чтобы сделать прохождение лабиринта более интересным. Константа WALL — тип блоков, из которых в лабиринте строятся стены, поэтому следует внимательно отнестись к выбору значения для этой константы. Некоторые блоки не могут служить стенами! Константа FLOOR — тип блоков, образующих основание (пол) лабиринта:

```
GAP = block.AIR.id
WALL = block.GOLD_BLOCK.id
FLOOR = block.GRASS.id
```

Не используйте тип SAND для блоков основания. Иначе, в зависимости от того, на чем построен лабиринт, некоторые блоки пола будут осыпаться под ногами персонажа.

5. Откройте файл с описанием лабиринта. Здесь используется константа FILENAME, и для вас не составит труда в будущем изменить имя файла, чтобы построить другой лабиринт на основе другого файла:

```
FILENAME = "maze1.csv"
f = open(FILENAME, "r")
```

6. Определите координаты персонажа и вычислите координаты нижнего угла лабиринта. Увеличение на 1 координат x и z гарантирует, что лабиринт не будет построен на голове персонажа. Вы можете изменить координату y, чтобы построить лабиринт в воздухе:

```
pos = mc.player.getTilePos()
ORIGIN_X = pos.x+1
ORIGIN_Y = pos.y
ORIGIN_Z = pos.z+1
```

7. После обработки каждой строки в файле координата z должна меняться, поэтому присвойте ей начальное значение — координату z нижнего угла лабиринта:

```
z = ORIGIN_Z
```

8. Организуйте обход в цикле всех строк файла. Во врезке «Углубляемся в код» вы найдете подробное описание действий функции `readlines()`. Цикл `for` фактически выполнит обход всех строк в файле. В каждой итерации цикла переменная `line` будет хранить очередную строку, прочитанную из файла:

```
for line in f.readlines():
```

9. Разбейте строку на части по запятым. Во врезке «Углубляемся в код» подробно рассказано, что делает функция `split()`. Не забудьте: все строки в программе, образующие тело цикла `for`, должны иметь по одному отступу:

```
  data = line.split(",")
```

10. Внимательно рассмотрите этот шаг: здесь начинается другой цикл `for`, который называется **вложенным**, потому что один цикл вкладывается в другой. Перед началом цикла программа должна сбросить координату *x* в начальное значение для каждого нового ряда лабиринта (это должно происходить за пределами цикла, строящего целый ряд):

```
  x = ORIGIN_X
  for cell in data:
```

11. Каждое число, прочитанное из файла, хранится в текстовом виде, поэтому числа, с которыми выполняется сравнение, в программе надо заключить в двойные кавычки `""`. Данная инструкция `if/else` определяет, строить стену или оставить пустое пространство, опираясь на очередное число из CSV-файла. Для `0` останется пустое пространство, для любого другого числа будет построена стена. Будьте внимательны при оформлении отступов: инструкция `if` должна иметь два отступа, потому что является частью цикла `for cell in data`, который, в свою очередь, является частью цикла `for line in f.readlines()`. Переменная `b` создана, чтобы слегка упростить программу и сделать ее короче. (Попробуйте переписать эту часть программы без переменной `b`, и вы поймете, что я имел в виду!)

```
    if cell == "0":
      b = GAP
    else:
      b = WALL
    mc.setBlock(x, ORIGIN_Y, z, b)
    mc.setBlock(x, ORIGIN_Y+1, z, b)
    mc.setBlock(x, ORIGIN_Y-1, z, FLOOR)
```

12. В конце цикла `for cell` надо увеличить координату *x*. Эта строка должна иметь столько же отступов, сколько предшествующая `mc.setBlock()`, потому что она тоже является частью цикла `for cell in data`.

```
    x = x + 1
```

13. В конце цикла, обрабатывающего строки из файла, нужно увеличить координату *z* (эта строка должна иметь один отступ, так как является частью цикла `for line in f.readlines()`).

```
  z = z + 1
```

Сохраните программу и еще раз проверьте все отступы. Если они оформлены неправильно, программа не будет корректно работать, и вы получите лабиринт очень странной формы!

Запустите программу: перед персонажем появится потрясающий и сложный лабиринт. Побродите по нему и постарайтесь выйти, не разрушая стены и не перепрыгивая через них. На рис. 5.2 показано, каким выглядит лабиринт глазами Стива.

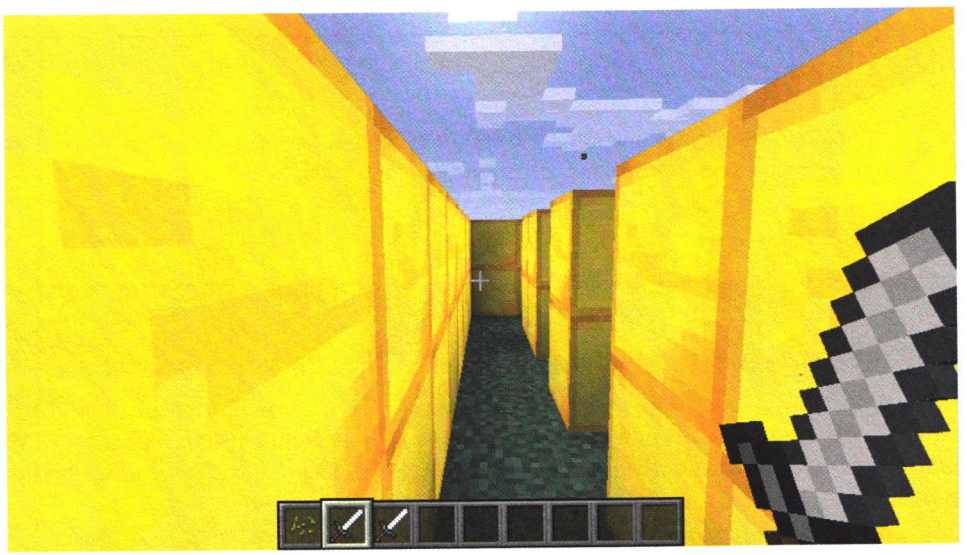


Рис. 5.2. Вид лабиринта в мире Minecraft глазами Стива

Если в лабиринте возникнут сложности, можете схитрить и «взлететь», чтобы посмотреть на него с высоты птичьего полета, как показано на рис. 5.3.

Рис. 5.3. Вид лабиринта в мире Minecraft с высоты птичьего полета

Вложенный цикл — это цикл внутри другого цикла. Первый цикл называется *внешним*, второй — *внутренним*. Вложенные циклы в языке Python оформляются с помощью дополнительного отступа. Циклы можно вкладывать друг в друга неограниченное количество раз.

Если вы случайно оставите несколько пустых строк в конце файла с данными, они будут прочитаны и обработаны программой на Python. К счастью, это не приведет к аварийному завершению программы, но в конце лабиринта может появиться несколько дополнительных блоков.

УГЛУБЛЯЕМСЯ В КОД

В предыдущих приключениях вы экспериментировали со списками в языке Python и узнали, что список — лишь коллекция из нуля или более нумерованных элементов. Многие встроенные функции языка Python возвращают данные в виде списка, две из которых вы встретили в программе по строительству лабиринта.

Первая функция была использована в следующей строке:

```
for line in f.readlines():
```

Напомню, что `f` — дескриптор, с помощью которого можно обращаться к открытому файлу. Очень удобно, что тип `file` в Python имеет встроенную функцию `readlines()`, которая читает каждую строку из файла и добавляет каждую строку в конец списка. Как уже было сказано выше, когда список передается в инструкцию цикла `for`, она выполняет обход всех элементов списка.

Фактически данная инструкция `for` сначала читает все строки из файла в список, а затем обходит их одну за другой. В каждой итерации цикла `for` переменная `line` (управляющая переменная цикла) хранит следующую строку с данными.

Второй важный момент в вашей программе — использование еще одного списка, но так, что вы его, скорее всего, не заметили. Это было в строке:

```
data = line.split(",")
```

Переменная `line` хранит полную строку текста, прочитанную из CSV-файла. Все строковые переменные имеют встроенную функцию с именем `split()`, которая преобразует строку в список. Аргумент `","` внутри круглых скобок сообщает функции `split`, по какому символу должно выполняться разбиение.

Представьте, что у вас есть переменная `line` с тремя словами, разделенными запятыми:

```
line = "one,two,three"
data = line.split(",")
```

В результате выполнения этих строк переменная `data` получит список с тремя элементами:

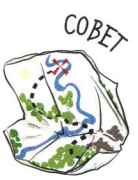
```
['one','two','three']
```

Спроектируйте собственные лабиринты с большим количеством поворотов, тупиков и циклов; сохраните их в CSV-файлах. Предложите друзьям найти выход из лабиринтов. Можете разбросать по ним сокровища (например, `DIAMOND_BLOCK`), собрав которые ваши друзья смогут доказать, что прошли весь лабиринт, или даже построить гигантский лабиринт, занимающий бóльшую часть мира Minecraft! Как бы вы обозначили позиции в файле лабиринта, где находятся сокровища?

Когда я проектировал программы для этой главы, пришлось не раз изменять программу строительства лабиринта, потому что иногда оказывалось сложно пройти то, что удалось построить. Например, что произойдет, если убрать из программы строку, которая создает пол? Попробуйте строить стены из блоков других типов, таких как `CACTUS` или `WATER_FLOWING`, и посмотрите, что будет. Я опробовал множество версий этой программы, прежде чем нашел типы блоков, из которых получились прекрасные стены и пол.

В этой программе скрыта ошибка: если последнее число в строке в файле будет равно нулю, программа строительства лабиринта в этом месте все равно создаст стену из золотого блока. Как вы думаете, почему так происходит? Попробуйте сами исправить ошибку. Подсказка: вы уже решали аналогичную проблему в программе *tipChat.py*.

В этом Приключении вы использовали для обработки CSV-файла функции `readlines()` и `split()`. Python — богатый язык программирования, он имеет множество встроенных функций. Вы можете рассмотреть встроенный модуль чтения файлов CSV, который легко импортировать инструкцией `import csv`, как более мощную альтернативу тому, что вы узнали здесь. Однако я люблю создавать программы постепенно, это помогает мне понять, как они работают. Поэтому я и использовал функции `readlines()` и `split()`.



Создание трехмерного принтера

Строительство лабиринтов — интересная задача, но теперь, познакомившись с основами обработки файлов данных, вы можете решать и более грандиозные задачи! Зачем останавливаться на строительстве одноэтажных сооружений из блоков только двух типов? Далее вы используете программу строительства алгоритмов как основу для создания собственного трехмерного принтера и трехмерного сканера, с помощью которых можно «копировать»

деревья, дома и все, что создается в мире Minecraft, а затем «напечатать» нажатием одной кнопки! В действительности это будет программа строительства из блоков, но мне захотелось назвать ее трехмерным принтером: порядный способ строительства напоминает то, как принтер печатает текст на листе бумаги или как устройство трехмерной печати слой за слоем создает копии.

Вы поднаторели в создании строительных программ, каждая новая программа получается больше предыдущей. Как и в предыдущем Приключении, где вы делали программу из функций, эта программа будет создаваться постепенно.

Прием, когда сначала создаются функции, реализующие особенности программы, которые затем «сшиваются» в одну большую программу, распространен среди профессиональных программистов. Он основан на представлении о том, что большая программа является простой коллекцией маленьких программ. Если общая архитектура программы не содержит ошибок, каждую функцию внутри можно заполнять постепенно, одновременно конструируя и тестируя программу. Я всегда предпочитал создавать программы, начиная с демонстрационной версии, чтобы, если кто-то подойдет и спросит: «Что за программу вы пишете?», — можно было показать действующие куски.

Подготовка вручную маленького объекта для трехмерной печати

Данные для программы строительства лабиринта хранятся в CSV-файле, где каждая строка представляет собой ряд блоков в мире Minecraft. Каждый столбец в этой строке (ограниченный запятыми) соответствует одному блоку в данном ряду блоков. Это двумерная структура, хранящая блок для каждой пары координат x и z в прямоугольной области. Лабиринты, которые строит ваша программа, в действительности являются двумерными, так как имеют лишь один уровень (этаж). Они просто строятся в трехмерном мире Minecraft из трехмерных блоков.

Чтобы построить трехмерное сооружение, необходимо задействовать координату y. Если представить трехмерный объект как множество слоев или срезов, программа строительства трехмерных объектов мало чем будет отличаться от программы строительства лабиринта. Все, что требуется сделать для куба $5 \times 5 \times 5$ блоков, — сохранить информацию о пяти слоях.

Однако здесь возникает проблема: программа на Python не знает, сколько будет строк, пока не обработает весь файл. Можно было бы принять конкретный размер, но лучше спроектировать более гибкий формат CSV-файлов, позволяющий определять объекты любых размеров.

Чтобы решить проблему, добавим некоторые **метаданные** в первую строку файла, которые описывают размеры объекта.

Метаданные — это данные о данных. Если данные в файле определяют типы блоков для строительства объекта, метаданными (данные о данных) могут быть, например, размеры объекта, его название и имя проектировщика. Как метаданные для ваших фотографий, хранящихся в компьютере, описывают дату и время съемки, название камеры, метаданные для трехмерного принтера несут полезную информацию о трехмерных объектах.

Теперь вам предстоит создать простой трехмерный объект — каменную камеру, на котором можно проверить работу программы трехмерного принтера. Вы можете загрузить готовый файл *CodeFiles.zip* с определением объекта, изображенного на рис. 5.4, с веб-сайта книги или ввести данные вручную, как уже это делали, создавая лабиринт.

Не забудьте добавить пустые строки между слоями объекта. Они помогут увидеть, где начинается каждый слой. Кроме того, программа на Python, которую вы напишете, ожидает встретить пустые строки и не будет работать, если их не окажется в файле.

Рис. 5.4. Каменная камера, описанная в CSV-файле

Используйте функцию копирования и вставки в редакторе при выполнении описанных ниже шагов, чтобы сэкономить время на вводе и уменьшить вероятность ввода ошибочного количества строк.

1. Создайте новый файл, выбрав в меню пункт File ▸ New File (Файл ▸ Создать файл), и сохраните его с именем *object1.csv*.
2. Введите в первую строку метаданные, описывающие размеры объекта. Ваш испытательный объект имеет размеры 5 × 5 × 5. Первое число определяет размер по оси *x* (ширину), второе — по оси *y* (высоту) и третье — по оси *z* (глубину):

   ```
   5,5,5
   ```

3. Введите первый слой объекта и не забудьте оставить пустую строку перед первой строкой. Это слой основания, поэтому во всех его позициях находятся блоки, а значит, во всех позициях должно стоять число 1:

   ```
   1,1,1,1,1
   1,1,1,1,1
   1,1,1,1,1
   1,1,1,1,1
   1,1,1,1,1
   ```

4. Сооружение представляет собой полый куб с открытой передней стенкой. Вставьте три слоя этого вида. Не забывайте оставлять пустую строку перед каждым слоем:

   ```
   1,1,1,1,1
   0,0,0,0,1
   0,0,0,0,1
   0,0,0,0,1
   1,1,1,1,1
   ```

5. Используйте функцию копирования и вставки в редакторе, чтобы добавить в файл еще два идентичных слоя с полостью, не забывая оставлять пустую строку перед каждым.
6. Наконец, добавьте сплошную крышу объекта. Для этого скопируйте и вставьте слой с данными из шага 3. И снова не забудьте оставить пустую строку перед первой строкой слоя.

   Сохраните файл, но не запускайте его — вам это не удастся, потому что это не программа на Python, а файл с данными. Его будет использовать программа, которую еще предстоит написать.

Создание трехмерного принтера

Теперь, когда у вас есть тестовые данные, можно приступать к созданию программы трехмерного принтера, которая воссоздаст макет каменной камеры в мире Minecraft. Эта программа напоминает программу строительства лабиринта, но в ней — три вложенных цикла, по одному для координат *x*, *y* и *z*.

Это, пожалуй, одна из самых подробных программ, которые вы писали и напишете еще, читая эту книгу. Вводите ее постепенно, с особым вниманием, и вы будете удивлены конечным результатом! Если программа не заработает сразу, внимательно проверьте все отступы и запомните: листинги всех программ в этой книге можно загрузить на веб-сайте книги: www.wiley.com/go/adventuresinminecraft2e.

1. Создайте новый файл, выбрав в меню пункт File ▸ New File (Файл ▸ Создать файл), и сохраните его с именем *print3D.py*, выбрав пункт меню File ▸ Save As (Файл ▸ Сохранить как).
2. Импортируйте необходимые модули:

```
import mcpi.minecraft as minecraft
import mcpi.block as block
```

3. Подключитесь к игре Minecraft:

```
mc = minecraft.Minecraft.create()
```

4. Создайте константу для хранения имени файла с данными. Ее легко изменить, если понадобится прочесть другие файлы с разными объектами:

```
FILENAME = "object1.csv"
```

5. Определите функцию `print3D()`. Она будет использоваться в заключительном проекте, поэтому постарайтесь ввести ее имя без ошибок:

```
def print3D(filename, originx, originy, originz):
```

6. Как в программе строительства лабиринта, откройте файл в режиме для чтения и прочитайте все строки в список Python:

```
  f = open(filename, "r")
  lines = f.readlines()
```

7. Первый элемент в списке имеет индекс 0. Это первая строка в файле, где хранятся метаданные. Разбейте строку на части с помощью функции `split()`. Затем сохраните три числа в переменных `sizex`, `sizey` и `sizez`. Здесь нужно использовать функцию `int()`, потому что из файла читаются текстовые строки, а вам нужны числа, которые можно использовать в дальнейших вычислениях:

```
  coords = lines[0].split(",")
  sizex = int(coords[0])
  sizey = int(coords[1])
  sizez = int(coords[2])
```

8. Сохраните в переменной `lineidx` индекс строки в обрабатываемом списке `lines[]`. Так как программа будет обрабатывать список `lines`, читая данные для разных слоев трехмерного объекта, ее нельзя использовать как управляющую переменную цикла:

```
  lineidx = 1
```

9. Первый цикл `for` выполняет обход слоев по вертикали. Первые несколько строк в файле определяют нижний слой, который надо построить. Сюда можно добавить вызов функции `postToChat`, чтобы ход строительства можно было наблюдать внутри Minecraft:

```
  for y in range(sizey):
    mc.postToChat(str(y))
```

10. Пропустите пустую строку, отделяющую слои (уровни), прибавив `1` к переменной `lineidx`. Не забывайте: пустые строки нужны только в файле, чтобы человеку было проще его читать, а программа их пропустит.

```
lineidx = lineidx + 1
```

11. Начните вложенный цикл `for`, который читает следующую строку из файла и разбивает ее по запятым. Будьте внимательны при оформлении отступов: инструкция `for` должна иметь два отступа (один как инструкция, принадлежащая функции, второй — как принадлежность циклу `for y`), а инструкции внутри цикла `for x` — три.

```
for x in range(sizex):
  line = lines[lineidx]
  lineidx = lineidx + 1
  data = line.split(",")
```

12. Теперь начните третий цикл `for`, который сканирует все блоки в только что прочитанной строке и создает их в мире Minecraft. Цикл `for z` должен иметь три отступа, его тело — четыре, поэтому будьте предельно внимательны при оформлении отступов. Если здесь допустить ошибку, вы получите объекты странной формы!

```
for z in range(sizez):
  blockid = int(data[z])
  mc.setBlock(originx+x, originy+y, originz+z, blockid)
```

13. Наконец, введите строки главной программы, не имеющие отступов. Они определяют координаты персонажа и вызывают функцию `print3D()` для строительства трехмерного объекта рядом с персонажем:

```
pos = mc.player.getTilePos()
print3D(FILENAME, pos.x+1, pos.y, pos.z+1)
```

Сохраните программу и запустите ее, чтобы посмотреть, как она работает. Перейдите в другое место в мире Minecraft и запустите программу повторно. Каждый раз при запуске программы рядом с персонажем должна появляться каменная камера. На рис. 5.5 показано, как легко построить много каменных камер.

Создайте еще несколько CSV-файлов с описаниями трехмерных объектов, измените константу `FILENAME` в программе *print3D.py* и запустите ее несколько раз, чтобы создать несколько копий объектов по всему миру Minecraft. По мере того как ваши объекты будут усложняться, придумайте несколько способов их проектирования перед вводом чисел в файл. Возможно, вы найдете способ рисования объектов в программе электронной таблицы. Или сможете купить большую коробку с деталями строительного конструктора и проектировать объекты в физическом мире, а затем послойно переносить их в CSV-файл.

Рис. 5.5. Трехмерная «печать» каменных камер в мире Minecraft

Создание сканера трехмерных конструкций

Трехмерный принтер — очень мощное устройство. Вы можете создать большую библиотеку CSV-файлов с разными объектами и затем строить их в мире Minecraft по мере необходимости с помощью программы *print3D.py*, подставляя разные имена файлов в константу `FILENAME`.

Однако сложно вручную вводить числа в CSV-файлы для определения больших и сложных объектов. Игра Minecraft — лучшая программа для строительства сложных сооружений. Раз так, почему бы не использовать ее для автоматического создания CSV-файлов? Здесь вы подойдете к дереву и обнимете его, а потом создадите идентичную копию, которую сможете воспроизводить в мире Minecraft раз за разом. К счастью, трехмерное сканирование выполняется почти так же, как трехмерная печать, только наоборот. Поэтому это будет не так сложно, как могло бы показаться.

1. Создайте новый файл, выбрав в меню пункт File ▸ New File (Файл ▸ Создать файл), и сохраните его с именем *scan3D.py*, выбрав пункт меню File ▸ Save As (Файл ▸ Сохранить как).
2. Импортируйте необходимые модули:

```
import mcpi.minecraft as minecraft
import mcpi.block as block
```

3. Подключитесь к игре Minecraft:

```
mc = minecraft.Minecraft.create()
```

4. Определите несколько констант для имени CSV-файла, куда будет сохранен результат сканирования, и размеры области для сканирования:

```
FILENAME = "tree.csv"
SIZEX = 5
SIZEY = 5
SIZEZ = 5
```

5. Определите функцию `scan3D()`. Она будет использована далее в программе, поэтому постарайтесь ввести ее имя без ошибок:

```
def scan3D(filename, originx, originy, originz):
```

6. Откройте файл, но на этот раз в режиме для записи, указав `"w"` в функции `open()`:

```
  f = open(filename, "w")
```

7. Первая строка должна содержать метаданные, поэтому запишите в нее размеры по осям *x*, *y* и *z*. Что означают символы `"\n"` в конце строки и для чего они нужны, объяснено ниже, во врезке «Углубляемся в код».

```
  f.write(str(SIZEX) + "," + str(SIZEY) + ","
          + str(SIZEZ) + "\n")
```

8. Программа сканера — это программа трехмерной печати, но перевернутая. Она так же использует три вложенных цикла. Следующий фрагмент программы приводится целиком, чтобы было проще правильно оформить отступы. Подробнее о том, как создаются строки со значениями, разделенными запятыми, рассказано ниже, во врезке «Углубляемся в код»:

```
  for y in range(SIZEY):
    f.write("\n")
    for x in range(SIZEX):
      line = ""
      for z in range(SIZEZ):
        blockid = mc.getBlock(originx+x,
                              originy+y,
                              originz+z)
        if line != "":
          line = line + ","
        line = line + str(blockid)
      f.write(line + "\n")
  f.close()
```

9. Наконец, введите строки главной программы, не имеющие отступов. Они определяют координаты персонажа, затем вычисляют координаты куба, в центре которого находится персонаж, и вызывают функцию `scan3D()` для сканирования всего объема в CSV-файл.

```
pos = mc.player.getTilePos()
scan3D(FILENAME, pos.x-(SIZEX/2), pos.y, pos.z-(SIZEZ/2))
```

Сохраните программу. Еще раз проверьте все отступы. Теперь подойдите к дереву, обнимите его (встаньте как можно ближе к стволу) и запустите программу *scan3D.py*. Что произошло?

А разве что-то должно было произойти? Напомню: эта программа сканирует объект в CSV-файл, поэтому вы должны заглянуть в него, чтобы увидеть, какие числа сохранены. Откройте файл *tree.csv*, выбрав в меню редактора пункт File ▸ Open (Файл ▸ Открыть). На рис. 5.6 показан фрагмент файла *tree.csv*, который получился у меня, после того как я обнял дерево на своем компьютере. Так как сканируемое пространство относительно невелико, вы можете получить лишь половину дерева, но вам ничто не мешает изменить константы `SIZE` и отсканировать бо́льший объем.

Поскольку процесс сканирования занимает какое-то время, вы можете увидеть, что файл отображается на вашем компьютере, прежде чем все данные будут сохранены в нем. Прежде чем открывать файл, подождите несколько секунд, чтобы убедиться, что процесс сканирования завершен.


```
5,5,5

0,0,0,0,0
0,0,17,0,0
0,0,0,0,0
0,0,0,0,0
78,78,78,78,78

0,0,0,0,0
0,0,17,0,0
0,0,0,0,0
0,0,0,0,0
0,0,0,0,0

18,18,18,18,18
18,18,17,18,18
18,18,18,18,18
18,18,18,18,18
0,0,0,0,0
```

Рис. 5.6. Содержимое CSV-файла после сканирования дерева

Напомню, что направление, в котором смотрит ваш персонаж, может не совпадать с направлением сканирования. Трехмерный сканер сканирует от позиции персонажа в сторону увеличения координат. То есть если персонаж находится в точке с координатами 0,0,0, трехмерный сканер сканирует пространство от этой точки по осям координат в сторону увеличения значений, например 4,4,4. Взгляните на рис. 2.1 в Приключении 2, где вы познакомились с координатами, чтобы понять, какие координаты соответствуют тем или иным направлениям.

Редактор IDLE обычно показывает только файлы с расширением *.py* в своих диалогах Open (Открыть) и Save (Сохранить). Чтобы увидеть в них свои файлы с расширением *.csv*, выберите в раскрывающемся списке Files of Type (Тип файлов) и в этих диалогах пункт All Files (Все файлы).

Теперь, когда у вас есть трехмерный сканер и трехмерный принтер, измените в программе *print3D.py* константу `FILENAME`, присвоив ей значение *tree.csv*, затем побродите по миру Minecraft, периодически запуская программу *print3D.py*. У вас должен получиться целый лес деревьев. Попробуйте воспроизвести деревья в воздухе и под водой, посмотрите, что из этого получится. Как быстро вы сможете создать целый лес таким способом?

УГЛУБЛЯЕМСЯ В КОД

В программе *scan3D.py*, в функции `write()`, используется специальный символ, о котором нужно рассказать отдельно:

```
f.write("\n")
```

Символ `\n` (произносится как «обратный слеш эн») — специальный непечатаемый символ, который язык Python позволяет использовать в двойных кавычках. Он означает «перейти на следующую строку». Символ `n` в этой комбинации происходит от английского «new line» (новая строка). Символ обратного слеша (обратной косой черты) используется, чтобы отличить эту букву `n` от обычной «n».

Формат файла предусматривает наличие пустых строк между слоями объекта, чтобы человеку было проще читать и редактировать CSV-файл с данными. Поэтому программа *scan3D.py* должна их вставлять. Вызов `f.write("\n")` как раз вставляет пустые строки.

Далее в программе *scan3D.py* находится интересный фрагмент, конструирующий строку в переменной `line`. Вот этот фрагмент:

```
for x in range(SIZEX):
  line = ""
  for z in range(SIZEZ):
    blockid = mc.getBlock(originx+x, originy+y, originz+z)
    if line != «»: # если line – не пустая строка
      line = line + ","
    line = line + str(blockid)
```

Выделенные **жирным шрифтом** строки — часть распространенного шаблона программирования, широко используемого для конструирования строк со значениями, разделенными запятыми. В начале цикла по `x` в переменную `line` записывается пустая строка. Всякий раз, когда цикл по `z` генерирует новое значение, сначала проверяется, не пустая ли строка в переменной `line`, и если нет, в ее конец добавляется запятая. В заключение в строку `line` добавляется идентификатор текущего блока `blockid`. Инструкция `if` необходима, чтобы предотвратить появление запятой в начале строки `line`, перед первым числом, что может сбить с толку программу *print3D.py*, когда она будет читать файл.

Создание копировального аппарата

Теперь у вас есть все строительные блоки, необходимые для создания собственного копировального аппарата, который станет предметом зависти ваших друзей. С новой программой в мире Minecraft можно материализовать волшебную камеру копировального аппарата, создавать внутри нее любые объекты, сохранять их в файлы, загружать из файлов в камеру, редактировать или создавать копии объектов по всему миру Minecraft. Наконец, вы сможете покинуть игровой мир и заставить камеру бесследно исчезнуть, оставив при себе результаты волшебства.

Как и в предыдущих Приключениях, создавая большую программу, вы будете использовать фрагменты из уже созданных вами программ. Так как эта программа обладает массой возможностей, вы добавите в нее систему меню, чтобы упростить управление.

Создание каркаса программы копировального аппарата

Первое, что нужно сделать, — создать каркас будущей программы. Сначала вы напишете несколько фиктивных функций, которые выводят свои имена, когда их вызывают, а затем постепенно наполните их программным кодом из существующих функций, написанных в других программах. Вы наверняка помните, что таким же способом конструировали программу поиска сокровищ в Приключении 4.

1. Создайте новый файл, выбрав в меню пункт File ▸ New File (Файл ▸ Создать файл), и сохраните его с именем *duplicator.py*.
2. Импортируйте необходимые модули:

```
import mcpi.minecraft as minecraft
import mcpi.block as block
import glob
import time
import random
```

3. Подключитесь к игре Minecraft:

```
mc = minecraft.Minecraft.create()
```

4. Определите несколько констант для хранения размера копировального аппарата. Не выбирайте слишком большие габариты, иначе аппарат будет слишком долго сканировать и воссоздавать объекты. Также определите начальные координаты копировальной камеры:

```
SIZEX = 10
SIZEY = 10
SIZEZ = 10
roomx = 1
roomy = 1
roomz = 1
```

5. Определите несколько фиктивных функций, которые потом будут делать все, что необходимо. Позднее вы заполните их программным кодом:

```
def buildRoom(x, y, z):
  print("buildRoom")

def demolishRoom():
  print("demolishRoom")

def cleanRoom():
  print("cleanRoom")

def listFiles():
  print("listFiles")

def scan3D(filename, originx, originy, originz):
  print("scan3D")

def print3D(filename, originx, originy, originz):
  print("print3D")
```

6. Фиктивная функция, реализующая меню, особенная, потому что в конце возвращает значение. Пока можно сгенерировать случайное число и **вернуть** его, чтобы иметь возможность проверить работу ранних версий программы. Скоро вы напишете полноценное меню. Фиктивная функция написана с целью проверить работу структуры программы; еще немного, и вы заполните ее настоящим программным кодом:

```
def menu():
  print("menu")
  time.sleep(1)
  return random.randint(1,7)
```

7. Напишите главный игровой цикл, отображающий меню и использующий функцию, соответствующую выбранному в нем числу. Подробнее о том, как используется переменная `anotherGo`, рассказано во врезке «Углубляемся в код»:

```
anotherGo = True
while anotherGo:
  choice = menu()
  if choice == 1:
    pos = mc.player.getTilePos()
    buildRoom(pos.x, pos.y, pos.z)
  elif choice == 2:
    listFiles()
  elif choice == 3:
    filename = raw_input("filename?")
    scan3D(filename, roomx+1, roomy+1, roomz+1)
  elif choice == 4:
    filename = raw_input("filename?")
    print3D(filename, roomx+1, roomy+1, roomz+1)
  elif choice == 5:
    scan3D("scantemp", roomx+1, roomy+1, roomz+1)
    pos = mc.player.getTilePos()
```

```
    print3D("scantemp", pos.x+1, pos.y, pos.z+1)
  elif choice == 6:
    cleanRoom()
  elif choice == 7:
    demolishRoom()
  elif choice == 8:
    anotherGo = False
```

Сохраните программу и запустите ее. Что вы видите? На рис. 5.7 показано, что произошло у нас, когда мы запустили эту программу на своем компьютере. Как видите, программа сообщает обо всех своих действиях — она выбрала случайный пункт в меню, а затем вызвала функцию, обрабатывающую его. Так как сейчас все функции в вашей программе просто выводят свои имена, вы увидите на экране лишь разные последовательности слов как на рис. 5.7, потому что функция `menu()` каждый раз делает случайный выбор.

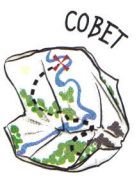
```
menu
filename?tree.csv
scan3D
menu
filename?tree.csv
scan3D
menu
filename?tree.csv
scan3D
menu
scan3D
print3D
menu
listFiles
menu
buildRoom
menu
scan3D
print3D
menu
filename?
```

Рис. 5.7. Результаты выполнения тестовой программы

Инструкция `return` дает возможность вернуть значение из функции в программу, которая использует функцию.

Например, когда требуется случайное число, вы вызываете функцию `random.randint()`. Она возвращает значение, которое вы сохраняете в переменной, например, так: `a = random.randint(1,100)`. Инструкция `return` в языке Python позволяет использовать тот же прием, чтобы вернуть значение из своей функции.

Функция `input()` читает строку текста, введенную с клавиатуры. Если добавить строку, заключенную в круглые скобки, эта строка появится на экране как приглашение к вводу, то есть инструкция `name = input("What is your name?")` задаст вопрос и вернет ответ.



Функция `input()` всегда возвращает введенный текст в виде строки. Если вам понадобится ввести число (например, в своей системе меню), используйте функцию `int()` для преобразования строки в число. Некоторые программисты на Python делают все в одной строке, например: `age = int(input("What is your age?"))`.

В этой книге все программы написаны на версии языка Python 3. Если у вас есть еще один компьютер, на нем можно установить версию Python 2. Две версии имеют несколько различий, и одно из них заключается в том, что в версии Python 2 следует использовать функцию `raw_input()` вместо `input()`. В этой книге используется Python 3.

УГЛУБЛЯЕМСЯ В КОД

Вы уже использовали **булевы**, или **логические**, переменные в предыдущих приключениях, а сейчас нужно пояснить, как используется переменная `anotherGo` в программе *duplicator.py*. Ниже приведены самые важные части программы:

```
anotherGo = True
while anotherGo:
  choice = menu()
  if choice == 8:
    anotherGo = False
```

Это распространенный способ организации циклов, которые выполняются хотя бы один раз, спрашивают, хотите ли вы продолжить, и в случае отрицательного ответа завершаются. В Python есть множество способов достичь той же цели, но использование булевой (логической) переменной, пожалуй, лучший вариант, наглядно демонстрирующий, как работает программа.

В языке Python есть инструкция `break`, прерывающая выполнение циклов, однако в этой книге мы не будем ее использовать. Вы можете провести свое исследование в интернете и узнать, как переписать этот цикл, чтобы использовать в нем инструкцию `break` и убрать булеву переменную.

Булева (**логическая**) переменная — переменная, которая может хранить только одно из двух значений: `True` или `False`. Свое название она получила в честь Джорджа Буля (George Boole) — английского математика, в XIX веке много работавшего с формальной логикой, исследования которого легли в основу теории, ставшей базой для создания современных компьютеров. Узнать больше о Джордже Буле можно здесь: https://ru.wikipedia.org/wiki/Буль,_Джордж.

Вывод меню

Система меню — очень удобная особенность и заслуживает быть добавленной в программы, когда требуется возможность выбора из нескольких вариантов. Данная система меню выводит все доступные варианты и затем ждет, когда будет введен номер варианта из допустимого диапазона. Если ввести число за его пределами, меню появится повторно и предоставит еще одну попытку выбора.

1. Измените функцию `menu` — замените ее содержимое следующим программным кодом (не забудьте правильно оформить отступы!):

```
def menu():
  while True:
    print("DUPLICATOR MENU")
    print(" 1. BUILD the duplicator room")
    print(" 2. LIST files")
    print(" 3. SCAN from duplicator room to file")
    print(" 4. LOAD from file into duplicator room")
    print(" 5. PRINT from duplicator room to player.pos")
    print(" 6. CLEAN the duplicator room")
    print(" 7. DEMOLISH the duplicator room")
    print(" 8. QUIT")
    choice = int(input("please choose: "))
    if choice < 1 or choice > 8:
      print("Sorry, please choose a number between 1 and 8")
    else:
      return choice
```

Сохраните программу и запустите ее. Чем отличается эта версия программы от предыдущей, с фиктивной функцией меню? На рис. 5.8 показано, как выглядит настоящее меню.

```
DUPLICATOR MENU
 1. BUILD the scanner room
 2. LIST files
 3. SCAN from scanner room to file
 4. LOAD from file into scanner room
 5. PRINT from scanner room to player.pos
 6. CLEAN the scanner room
 7. DEMOLISH the scanner room
 8. QUIT
please choose:
```

Рис. 5.8. Система меню

Написав каркас программы с системой меню, заполнив фиктивные функции одну за другой и повторно протестировав работу программы, вы поступите как настоящий инженер-программист, разрабатывающий компьютерные программы. Это хорошая практика — писать программы небольшими фрагментами, внося изменения и проверяя их работу, пока не получится законченная программа. Кроме того, такой подход позволяет всего за пару минут продемонстрировать программу любому заинтересовавшемуся человеку, которому вы не сможете отказать!

Создание копировальной камеры

Копировальная камера строится из стекла и без передней стенки, чтобы персонаж мог легко зайти в нее и добавить или удалить блоки.

Замените содержимое функции `buildRoom()` следующим программным кодом. Будьте внимательны при вводе длинных строк, где используется функция `setBlocks()`, и обратите внимание на то, что на этот раз мы не использовали стрелку ↩, потому что Python позволя-

ет разбивать длинные строки на несколько строк (подробнее о том, в каких ситуациях можно переносить длинные строки, а в каких — нет, рассказано во врезке «Углубляемся в код»):

```
def buildRoom(x, y, z):
  global roomx, roomy, roomz
  roomx = x
  roomy = y
  roomz = z
  mc.setBlocks(roomx, roomy, roomz,
        roomx+SIZEX+1, roomy+SIZEY+2, roomz+SIZEZ+1,
        block.GLASS.id)
  mc.setBlocks(roomx+1, roomy+1, roomz,
        roomx+SIZEX+1, roomy+SIZEY+1, roomz+SIZEZ+1,
        block.AIR.id)
```

Сохраните программу и запустите ее. Проверьте работу варианта 1 в меню, чтобы убедить ся, что он создает копировальную камеру. Перейдите в другое место в мире Minecraft и снова выберите вариант 1, чтобы увидеть, что из этого получится. На рис. 5.9 показана копировальная камера сразу после ее постройки.

Уничтожение копировальной камеры

Если запустить программу копировального аппарата много раз, в мире Minecraft появится множество копировальных камер, а функционировать будет лишь последняя. Спустя какое-то время в вашем мире окажется столько копировальных камер, что вы запутаетесь, какая из них действующая! Для решения этой проблемы нужно добавить в программу возможность уничтожить ставшую ненужной копировальную камеру, чтобы мир Minecraft не захламлялся.

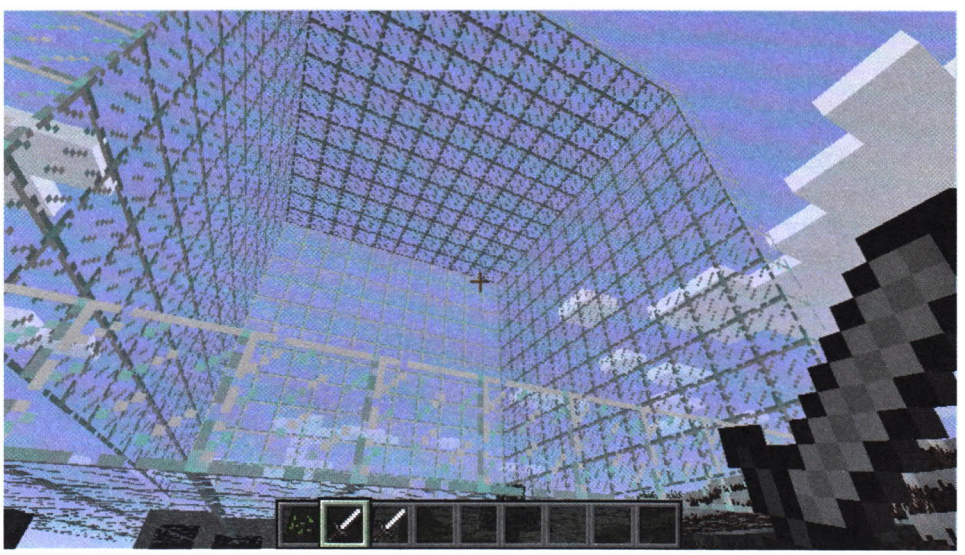


Рис. 5.9. Копировальная камера, построенная прямо перед персонажем

УГЛУБЛЯЕМСЯ В КОД

Отступы (сделанные с помощью пробелов или табуляторов) в языке Python используются, чтобы определить группы инструкций. Всякий раз, когда в программе используются циклы, инструкции `if`/`else` или функции, нужно добавлять отступы к сгруппированным инструкциям. Язык Python необычен как раз тем, что для группировки инструкций в блоки использует отступы. Во многих других языках, таких как C и C++, для этого используются специальные символы, например `{` и `}`. Поэтому при программировании на языке Python приходится уделять особое внимание оформлению отступов, иначе программа работает неправильно.

В большинстве случаев язык Python не позволяет переносить длинные строки. Именно поэтому в данной книге часто можно видеть стрелку ↩ в листингах, указывающую на то, что длинная строка не должна переноситься. Однако в Python есть два других способа справиться с длинными строками.

Во-первых, можно использовать символ продолжения строки: если последним в строке поставить символ обратного слеша (`\`), вы можете продолжить программную инструкцию на следующей строке.

```
a = 1
if a == 1 or \
a == 2:
  print("yes")
```

Второй способ, который иногда используется для переноса длинных строк, — выполнять перенос в точках, где интерпретатор Python сможет понять, что инструкция не закончилась. Например, когда определяется начальное значение списка или используется функция, инструкцию можно перенести — если Python поймет, что далее следует код. Ниже даны два примера с переносом строки. Так, открывающая скобка явно показывает, что строка не завершена, пока не встретится закрывающая скобка:

```
names = ["David",
"Steve",
"Joan",
"Joanne"
]

mc.setBlocks(x1, y1, z1,
x2, y2, z2, block.AIR.id)
```

Уничтожение копировальной камеры выполняется так же, как очистка пространства в программе *clearSpace.py* из Приключения 3. Вам нужно лишь знать внешние координаты камеры. Поскольку камера имеет толщину стен в один блок, ее внешние размеры на единицу больше внутреннего рабочего пространства, определяемого константами `SIZEX`, `SIZEY` и `SIZEZ`; вы легко их вычислите, использовав простую арифметику.

Измените функцию `demolishRoom()`, как показано ниже.

```
def demolishRoom():
  mc.setBlocks(roomx, roomy, roomz,
    roomx+SIZEX+1, roomy+SIZEY+1, roomz+SIZEZ+1,
    block.AIR.id)
```

Сохраните и запустите программу. Теперь вы легко построите копировальную камеру и уничтожите ее. Достаточно выбрать вариант `1` для постройки и вариант `7` для уничтожения.

Сканирование объектов в копировальной камере

Вы уже написали эту часть программы в программе *scan3D.py*, поэтому просто вставьте готовое тело функции, внеся небольшие изменения:

1. Замените функцию `scan3D()` следующим программным кодом. Этот код почти идентичен коду в программе *scan3D.py* — в него добавлены строки, выделенные жирно, чтобы отразить ход сканирования в чате Minecraft. Сканирование в камере большого объема может длиться долго, поэтому хорошо иметь перед глазами индикацию, показывающую, как идет процесс. Вы можете скопировать функцию из предыдущей программы и сэкономить время на вводе:

```
def scan3D(filename, originx, originy, originz):
  f = open(filename, "w")
  f.write(str(SIZEX) + "," + str(SIZEY) + "," + ↩
    str(SIZEZ) + "\n")
  for y in range(SIZEY):
    mc.postToChat(«scan:» + str(y))
    f.write("\n")
    for x in range(SIZEX):
      line = ""
      for z in range(SIZEZ):
        blockid = mc.getBlock(originx+x, originy+y, ↩
          originz+z)
        if line != "":
          line = line + ","
        line = line + str(blockid)
      f.write(line + "\n")
  f.close()
```

2. Сохраните программу и проверьте, как она работает. Для этого зайдите в копировальную камеру, постройте что-нибудь, а затем выберите вариант `3` в меню. Откройте файл, созданный в результате, и проверьте, правильно ли выполнено сканирование. На рис. 5.10 показан фрагмент файла, полученного в результате сканирования. Обратите внимание на большое количество нулей — это объясняется тем, что блоки типа `AIR` тоже были зафиксированы сканером.

```
10,10,10

0,0,0,0,0,0,0,0,0,0
0,0,0,0,0,0,0,0,0,0
0,0,0,0,0,0,0,0,0,0
0,0,0,0,0,0,0,0,0,0
0,0,0,0,1,0,0,1,1,0
0,0,0,0,1,1,1,1,1,0
0,0,0,0,0,0,0,0,0,0
0,0,0,0,0,0,0,0,0,0
0,0,0,0,0,0,0,0,0,0
0,0,0,0,0,0,0,0,0,0

0,0,0,0,0,0,0,0,0,0
0,0,0,0,0,0,0,0,0,0
0,0,0,0,0,0,0,0,0,0
0,0,0,0,0,0,0,0,0,0
0,0,0,0,0,0,0,0,0,0
0,0,0,0,1,0,0,0,0,0
```

Рис. 5.10. Содержимое файла, полученного в результате сканирования

Очистка копировальной камеры

Представьте себе генеральную уборку. Полезно иметь возможность очистить камеру, чтобы начать с нуля. Для этого можно было бы просто уничтожить камеру и вновь ее создать, но гораздо проще добавить функцию очистки. Она будет отличаться от функции `demolishRoom()` только используемыми координатами:

1. Измените функцию `cleanRoom()`, чтобы она выглядела, как показано ниже. Обратите внимание: все начальные координаты в этой функции больше координат границ комнаты, чем в функции `demolishRoom()`, а конечные координаты — меньше. Благодаря этому уничтожено будет лишь то, что находится внутри камеры, а ее стены останутся целыми:

```
def cleanRoom():
  mc.setBlocks(roomx+1, roomy+1, roomz+1,
    roomx+SIZEX, roomy+SIZEY, roomz+SIZEZ, block.AIR.id)
```

2. Сохраните и запустите программу. Постройте что-нибудь внутри камеры и затем выберите вариант 6, чтобы проверить, сможете ли вы быстро очистить камеру.

Воспроизведение объектов в копировальной камере

Написать функцию воспроизведения (создания копии) объекта, находящегося в копировальной камере, очень просто: у вас уже есть готовая программа *print3D.py*, которая делает это. Ниже мы повторим программный код функции, чтобы вы увидели ее целиком:

```
def print3D(filename, originx, originy, originz):
  f = open(filename, "r")
  lines = f.readlines()
  coords = lines[0].split(",")
  sizex = int(coords[0])
  sizey = int(coords[1])
  sizez = int(coords[2])
  lineidx = 1
```

```
    for y in range(sizey):
      mc.postToChat("print:" + str(y))
      lineidx = lineidx + 1
      for x in range(sizex):
        line = lines[lineidx]
        lineidx = lineidx + 1
        data = line.split(",")
        for z in range(sizez):
          blockid = int(data[z])
          mc.setBlock(originx+x, originy+y, originz+z, ↩
            blockid)
```

Сохраните и запустите программу. Постройте что-нибудь в копировальной камере, затем перейдите в другое место Minecraft и выберите в меню вариант 5. Содержимое камеры будет отсканировано и воспроизведено прямо перед персонажем. Попробуйте снова перейти в другое место мира Minecraft и воспроизвести объект еще несколько раз. Воспроизведите копию объекта в воздухе и под водой, чтобы посмотреть, что получится. На рис. 5.11 показаны результаты работы этой программы.

Рис. 5.11. Каменная постройка, находящаяся в копировальной камере, воспроизводится рядом с персонажем

Представление файлов

Напоследок вам предстоит написать небольшую вспомогательную функцию, перечисляющую все файлы с расширением *.csv*. Можно было бы просто воспользоваться Проводником, но хорошо добавить такую возможность в программу, чтобы все необходимое всегда имелось под рукой:

1. Измените функцию `listFiles()`, как показано ниже, и используйте в ней функцию `glob.glob()`, чтобы получить список всех файлов и вывести его. Подробное описание этой функции — во врезке «Углубляемся в код».

```
def listFiles():
  print("\nFILES:")
  files = glob.glob("*.csv")
  for filename in files:
    print(filename)
  print("\n")
```

2. Сохраните и запустите программу. Отсканируйте несколько объектов в CSV-файлы, затем выберите вариант 2 в меню и проверьте, все ли файлы присутствуют в списке. На рис. 5.12 перечислены файлы, которые мы создали, тестируя программу на своем компьютере.

```
DUPLICATOR MENU
 1. BUILD the scanner room
 2. LIST files
 3. SCAN from scanner room to file
 4. LOAD from file into scanner room
 5. PRINT from scanner room to player.pos
 6. CLEAN the scanner room
 7. DEMOLISH the scanner room
 8. QUIT
please choose: 2

FILES:
blocks.csv
maze1.csv
object1.csv
tree.csv


DUPLICATOR MENU
 1. BUILD the scanner room
 2. LIST files
 3. SCAN from scanner room to file
 4. LOAD from file into scanner room
 5. PRINT from scanner room to player.pos
 6. CLEAN the scanner room
 7. DEMOLISH the scanner room
 8. QUIT
please choose:
```

Рис. 5.12. Список созданных CSV-файлов

УГЛУБЛЯЕМСЯ В КОД

`glob.glob()` — какое забавное имя функции! Настолько, что его нужно ввести дважды.

Имя `glob` используется два раза потому, что первое имя — имя модуля (модуля `glob`), который был импортирован в начале программы, а второе — имя функции `glob` внутри модуля `glob`.

Что делает функция `glob` и почему она так называется?

Данное имя является сокращением английского словосочетания global command (глобальная команда) и корнями уходит в раннюю эпоху развития операционной системы UNIX. Познакомиться ближе с историей появления `glob` можно в Википедии (интернет-энциклопедии), на странице: http://en.wikipedia.org/wiki/Glob_(programming).

Эта функция просто создает список файлов, имена которых соответствуют указанному **шаблону**. Rulf программа вызывает `glob.glob("*.csv")`, Python просматривает текущий каталог в файловой системе компьютера и создает список всех файлов, имена которых заканчиваются на `".csv"`.

То есть если в папке *MyAdventures* имеются файлы *one.csv*, *two.csv* и *three.csv*, вызов функции `glob.glob("*.csv")` вернет список Python, который будет выглядеть так:

```
['one.csv', 'two.csv', 'three.csv']
```

А теперь вспомните, что цикл `for` выполняет итерации по всем элементам списка. Это объясняет, почему далее в программе стоят следующие строки:

```
for name in glob.glob("*.csv")
  print(name)
```

Шаблон — это специальная строка, которая используется для выбора множества похожих имен или слов. Он как джокер — карта в игральной колоде, способная исполнять любую роль, представлять все, что требуется.

В Python шаблоны часто используют для выбора множества файлов с похожими именами, например все CSV-файлы. Для этого можно вызвать `glob.glob("*.csv")`, где символ `*` отмечает позицию джокера, и `glob.glob()` отберет все файлы в файловой системе, имена которых заканчиваются на `.csv`.

Если в программе *duplicator.py* ввести имя несуществующего файла, она остановится с ошибкой и оставит копировальную камеру в мире Minecraft. Сделайте свою программу более надежной, найдя в интернете способ определить, что файл не существует, выполнить проверку в программе и не допустить ее аварийного завершения. Расширьте данные, которые записываются в файл, чтобы дополнительные данные также сохранялись в файле и извлекались из него. Это позволяет хранить блоки `WOOL` разных цветов, а также просматривать и создавать разноцветные творения.

Краткая справочная таблица

Команда	Описание

`f = open("data.txt", "r")`
`tips = f.readlines()`
`f.close()`
`for t in tips:`
`  print(t)`
	Чтение строк из файлов

Краткая справочная таблица	
Команда	**Описание**

`f = open("scores.txt", "w")`
`f.write("Victoria:26000\n")`
`f.write("Amanda:10000\n")`
`f.write("Ria:32768\n")`
`f.close()`
	Запись строк в файл

`import glob`
`names = glob.glob("*.csv")`
`for n in names:`
`  print(n)`
	Получение списка файлов с именами, соответствующими шаблону

`a = "\n\n hello \n\n"`
`a = a.strip()`
`print(a)`
	Удаление нежелательных пробельных символов по краям строк

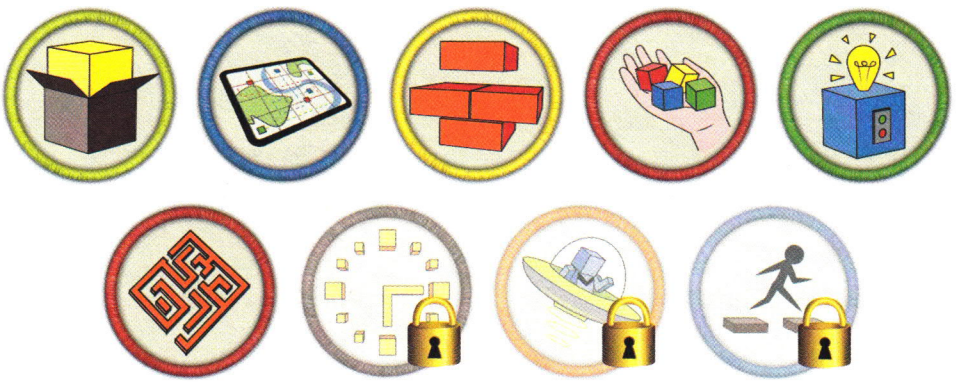
Дополнительные приключения с файлами данных

В этом Приключении вы научились читать данные из файлов и записывать их в файлы. Эти знания открывают перед вами безграничные возможности по сохранению фрагментов мира Minecraft и по импортированию больших объемов данных из других источников, таких как веб-сайты. Вы построили собственные трехмерные лабиринты с множеством поворотов, тупиков и циклов, а закончили созданием полноценного трехмерного принтера и сканера со своей системой меню. Способ создания такого меню пригодится во многих других программах!

В интернете есть много источников оперативной информации. Один из них, который мы нашли во время поиска информации для этой книги, — данные о парковке Ноттингемского муниципалитета: http://data.nottinghamtravelwise.org.uk/parking.xml. Эти данные обновляются каждые пять минут и показывают наличие свободных мест на автомобильных парковках. Я написал программу на языке Python, которая обрабатывает данные и выводит сводную информацию, и выложил ее в своем репозитории github: https://github.com/whaleygeek/pyparsers. Сможете ли вы воспользоваться ею, чтобы написать игру для Minecraft, которая строит парковки с автомобилями внутри мира Minecraft? Ваша игра могла бы расставлять парковки по всему миру Minecraft, а перед вами стояла бы задача за ограниченное время найти на парковке свободное место для своего автомобиля.

Все, что вы хотели бы знать о трехмерных лабиринтах, рассказано на этой замечательной странице: http://www.astrolog.org/labyrnth/algrithm.htm. Здесь вы найдете множество примеров трехмерных многоуровневых лабиринтов, хранящихся в обычных текстовых файлах. Загляните на этот сайт. Возможно, там вы найдете файл с данными для многоуровневого ла-

бирinta и измените программу строительства, использовав приемы, с которыми познакомились при создании трехмерного копировального аппарата, чтобы строить большие многоуровневые лабиринты. Можете воспользоваться и некоторыми подсказками и написать программу на Python, которая будет автоматически генерировать случайные лабиринты.



Новое достижение: нарушитель законов физики, способный материализовать и дематериализовать большие объекты в мире Minecraft одним нажатием клавиши.

В СЛЕДУЮЩЕМ ПРИКЛЮЧЕНИИ

В Приключении 6 вы узнаете, как строить сложные двух- и трехмерные объекты в программах на языке Python. Научитесь отмечать время с помощью часов Minecraft, строить многоугольники и другие многосторонние фигуры с помощью нескольких строчек кода на Python — и отправитесь в захватывающую экскурсию к пирамидам!

Приключение 6
Строительство двух- и трехмерных структур

ОДНА ИЗ замечательных сторон программирования для Minecraft — возможность отображения на двумерном экране сооружений, созданных в виртуальном, трехмерном мире. Вы можете обойти их кругом, заглянуть внутрь и даже взорвать, если этого захочется! Используя идеи отображения трехмерных объектов на двумерном экране, вы можете программировать трехмерные объекты в Minecraft, превращая обыденное в экстраординарное.

В этом Приключении вы узнаете, как пользоваться модулем `minecraftstuff` для создания двумерных линий и фигур, которые с помощью нехитрых арифметических вычислений можно превратить в часы — настолько большие, что можно вставать на их стрелки и кататься по кругу (рис. 6.1). Овладев приемами создания двумерных фигур, вы узнаете, как их объединять в огромные трехмерные структуры.

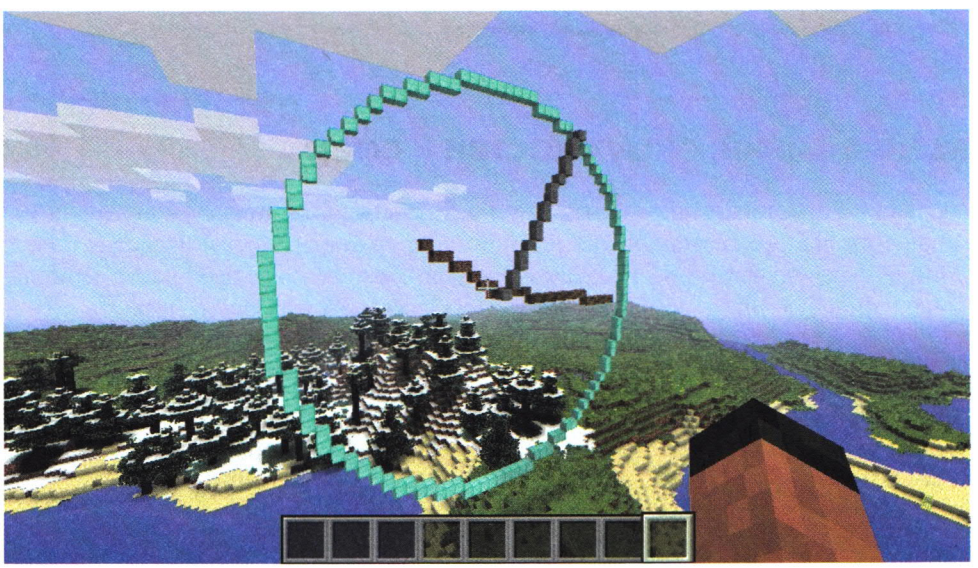


Рис. 6.1. Огромные часы в мире Minecraft

Модуль minecraftstuff

`Minecraftstuff` — это модуль расширения прикладного программного интерфейса Minecraft API, который был написан специально для книги «*Minecraft. Программируй свой мир*» и который содержит весь программный код, необходимый для создания геометрических фигур и управления трехмерными объектами. Он включает в себя множество функций под именем `MinecraftDrawing`, с помощью которых можно создавать линии, окружности, сферы и другие фигуры. Так как сложный код находится в отдельном **модуле**, вам будет проще писать свои программы и вы будете лучше их понимать. Модули позволяют разделить программу на части. Ведь когда программы становятся громоздкими, их сложно читать, еще сложнее — понимать, и приходится тратить больше времени на поиск ошибок.

Модуль `minecraftstuff` входит в начальный пакет для книги «*Minecraft. Программируй свой мир*», который вы установили в Приключении 1, его можно импортировать в программу для Minecraft как модули `minecraft` и `block`.

У модуля `minecraftstuff` гораздо больше возможностей, чем описано в книге. Поэтому обязательно ознакомьтесь с документацией (https://minecraft-stuff.readthedocs.io) и поэкспериментируйте с ним.

Если вы не используете начальный набор, то есть инструкции по установке, вам нужно просто изменить способ импорта модуля `minecraftstuff`, так как для удобства он включен в папку *mcpi* начального набора.

Создание линий, окружностей и сфер

Из маленьких и простых деталей можно построить большую и сложную конструкцию. В этом Приключении вы будете использовать линии, окружности, сферы и другие фигуры для создания в мире Minecraft по-настоящему больших структур. В этой части Приключения вы создадите новую программу, импортируете необходимые модули и настроите модули `minecraft` и `minecraftdrawing`, а потом будете использовать функции из них для создания линий, окружностей, сфер.

Запустите Minecraft и Python IDLE. У вас уже должна быть небольшая практика с запуском, но вы можете обратиться к Приключению 1, если нужно освежить свои знания.

1. Откройте редактор IDLE и создайте новый файл. Сохраните файл с именем *LinesCirclesAndSpheres.py* в папке *MyAdventures*.
2. Импортируйте модули `minecraft`, `block`, `minecraftstuff` и `time`, добавив в программу следующие строки:

```
import mcpi.minecraft as minecraft
import mcpi.block as block
import mcpi.minecraftstuff as minecraftstuff
import time
```

3. Подключитесь к игре Minecraft:

```
mc = minecraft.Minecraft.create()
```

4. Создайте с помощью модуля `minecraftstuff` объект `MinecraftDrawing`:

```
mcdrawing = minecraftstuff.MinecraftDrawing(mc)
```

УГЛУБЛЯЕМСЯ В КОД

`MinecraftDrawing` — это класс в модуле `minecraftstuff`. Классы являются частью специализированной методики программирования, которая называется объектно-ориентированной и позволяет группировать данные и функции для их обработки в одно целое. В данном случае — функции рисования в Minecraft и данные, чтобы их было проще использовать. Используя класс и давая ему имя (например, `mcdrawing`), вы создаете объект. Такая операция называется созданием экземпляра класса. Объект напоминает переменную, но может хранить не только значения (или данные), но и функции!

Объектно-ориентированное программирование (ООП) — очень сложная тема, которой посвящены целые тома с описанием, что это такое и как эффективно использовать данную методику. Тем не менее довольно простое введение в использование классов на языке Python есть по адресу https://ru.wikipedia.org/wiki/Объектно-ориентированное_программирование_на _Python.

Вы можете заглянуть в модуль `minecraftstuff`, открыв файл *minecraftstuff.py* в папке *MyAdventures/mcpi* с помощью IDLE.

Создание линий

Объект `MinecraftDrawing` имеет функцию `drawLine()`, принимающую две позиции (x, y, z) и тип блока в параметрах, которая создает линию из блоков, соединяющую эти две позиции. В точности как функция `setBlocks()`, с которой вы познакомились в Приключении 3.

Если для работы функции нужна информация, — например, для функции `setBlock()` нужны координаты *x*, *y*, *z* и тип блока, — она принимает значения, называемые **параметрами**. Когда программа использует функцию, говорят, что она **вызывает** ее и **передает** ей параметры.

В следующем коде используется функция `drawLine` для создания линии из блоков, как показано на рис. 6.2:

```
drawLine(x1, y1, z1, x2, y2, z2, blockType, blockData)
```

Теперь добавьте в программу вызовы функции `drawLine()`, чтобы создать три линии в мире Minecraft: вертикальную, горизонтальную и по диагонали от местоположения персонажа. Для этого добавьте в программу *LinesCirclesAndSpheres.py* следующий код:

1. Определите координаты персонажа:

```
pos = mc.player.getTilePos()
```

2. Чтобы построить вертикальную линию высотой в 20 блоков, вверх от позиции персонажа введите:

```
mcdrawing.drawLine(pos.x, pos.y, pos.z,
                   pos.x, pos.y + 20, pos.z,
                   block.WOOL.id, 1)
```

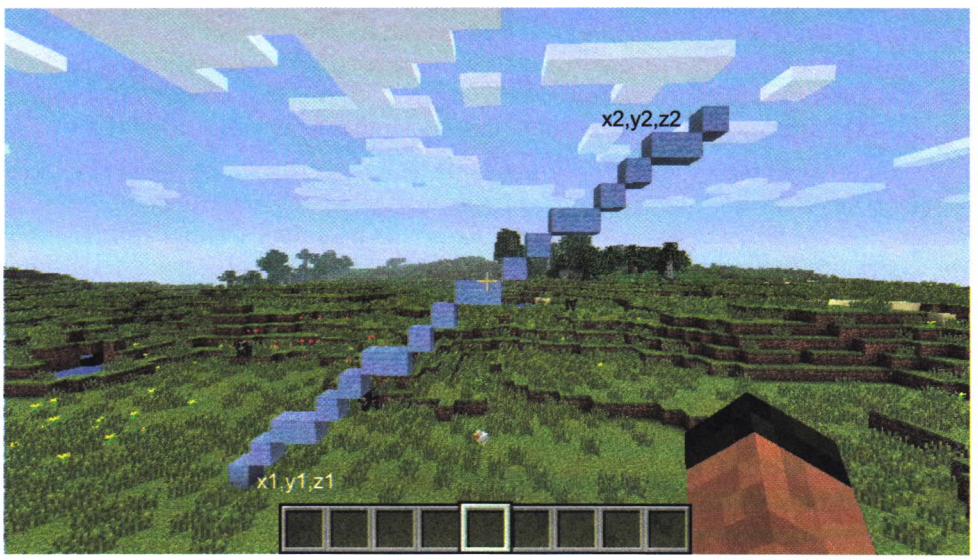


Рис. 6.2. Функция drawLine() создает линию из блоков между двумя точками с координатами x, y, z

3. Теперь постройте горизонтальную линию длиной в 20 блоков, вправо от позиции персонажа:

```
mcdrawing.drawLine(pos.x, pos.y, pos.z,
                   pos.x + 20, pos.y, pos.z,
                   block.WOOL.id, 2)
```

4. Далее постройте диагональную линию длиной в 20 блоков, вправо и вверх от позиции персонажа:

```
mcdrawing.drawLine(pos.x, pos.y, pos.z,
                   pos.x + 20, pos.y + 20, pos.z,
                   block.WOOL.id, 3)
```

5. Добавьте задержку времени, чтобы между каждым разделом был промежуток. Он позволяет увидеть, что происходит:

```
time.sleep(5)
```

6. Сохраните и запустите программу. У вас должны получиться три линии из блоков шерсти разного цвета: одна будет тянуться вертикально вверх от позиции персонажа, одна — горизонтально вправо и одна — по диагонали между ними.

УГЛУБЛЯЕМСЯ В КОД

Для строительства линий функция `drawLine()` использует алгоритм Брезенхэма (Bresenham). Чтобы больше о нем узнать, загляните на страничку https://ru.wikipedia.org/wiki/Алгоритм_Брезенхэма.

Создание окружностей

Нет причин останавливаться на линиях — с помощью объекта `MinecraftDrawing` можно создавать окружности, используя функцию `drawCircle()`, которой надо передать координаты центра окружности, радиус и тип блоков. Вы можете создать окружность, используя следующий код для вызова функции `drawCircle`:

```
drawCircle(x, y, z, radius, blockType, blockData)
```

Чтобы построить окружность, изображенную на рис. 6.3, добавьте следующие строки в конец программы *LinesCirclesAndSpheres.py*:

1. Сначала определите координаты персонажа:

```
pos = mc.player.getTilePos()
```

2. Теперь постройте окружность с центром на 20 блоков выше позиции персонажа и с радиусом в 20 блоков:

```
mcdrawing.drawCircle(pos.x, pos.y + 20, pos.z, 20,
                     block.WOOL.id, 4)
```

3. Добавьте задержку, чтобы можно было отойти в другое место и увидеть, что получилось:

```
time.sleep(5)
```

4. Сохраните и запустите программу.

Сначала программа построит линии, затем у вас будет 5 секунд, чтобы перейти в другое место, после чего программа построит окружность.

УГЛУБЛЯЕМСЯ В КОД

Для строительства окружностей функция `drawCircle()` использует алгоритм срединной точки. Он является адаптацией алгоритма Брезенхэма (Bresenham). Дополнительную информацию об этом алгоритме можно найти на странице https://ru.wikipedia.org/wiki/Алгоритм_Брезенхэма. Помимо создания вертикальных кругов, вы можете использовать функцию `drawHorizontalCircle ()` для создания кругов, которые лежат в плоскости.

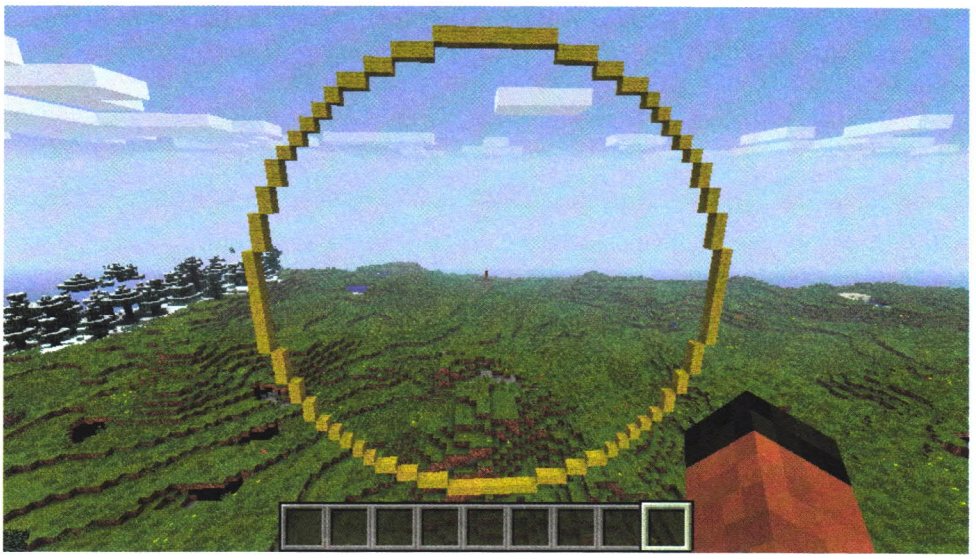


Рис. 6.3. Функция drawCircle создает окружность с центром в позиции (x, y, z) и заданным радиусом

Создание сфер

Функция `drawSphere()` похожа на функцию `drawCircle()` тем, что принимает координаты центра сферы, радиус и тип блоков, а также создает сферу с помощью следующего кода для вызова функции `drawSphere()`:

```
drawSphere(x, y, z, radius, blockType, blockData)
```

Чтобы создать сферу, изображенную на рис. 6.4, добавьте следующие строки в конец программы *LinesCirclesAndSpheres.py*:

1. Определите координаты персонажа:

```
pos = mc.player.getTilePos()
```

2. Чтобы построить сферу с центром выше персонажа на 20 блоков и с радиусом в 15 блоков, введите:

```
mcdrawing.drawSphere(pos.x, pos.y + 20, pos.z, 15,
                     block.WOOL.id, 5)
```

3. Сохраните и запустите программу.

Программа снова построит линии и окружность, каждый раз давая по 5 секунд, чтобы отвести персонажа в сторону, и затем построит сферу.

Полный программный код, создающий линии, окружности и сферы можно найти на вебсайте книги www.wiley.com/go/adventuresinminecraft2e.

Рис. 6.4. Функция drawSphere создает сферу с центром в позиции (x, y, z) и заданным радиусом

Сферы отлично подходят для имитации взрывов в Minecraft. Создав сферу из блоков `AIR`, можно удалить другие блоки вокруг центра сферы и получить «дыру» в мире.

Теперь, когда вы знаете, насколько просто программировать создание простых фигур, попробуйте сделать собственный трехмерный художественный шедевр из линий, окружностей и сфер.

Создание часов

Зная, как строить окружности и линии, вы наверняка захотите научиться строить часы, изображенные на рис. 6.1. Их циферблат — это большая окружность, а стрелки — линии. Поговорим о самой сложной части программы: как правильно расположить стрелки и заставить их двигаться.

На сайте книги есть видеоурок, где показано, как создать часы в мире Minecraft. Для этого посетите веб-сайт www.wiley.com/go/adventuresinminecraft2e и выберите видеоурок для Приключения 6.

В этой части Приключения вы будете использовать формулы **тригонометрии**, чтобы определить, куда должны указывать стрелки, преобразуя углы их поворота в координаты x и y на циферблате часов (рис. 6.5). Вы сымитируете движение стрелок, сначала создав соответствующие отрезки из блоков, затем удалив их и заменив блоками типа `AIR` и снова создав отрезки в новой позиции.

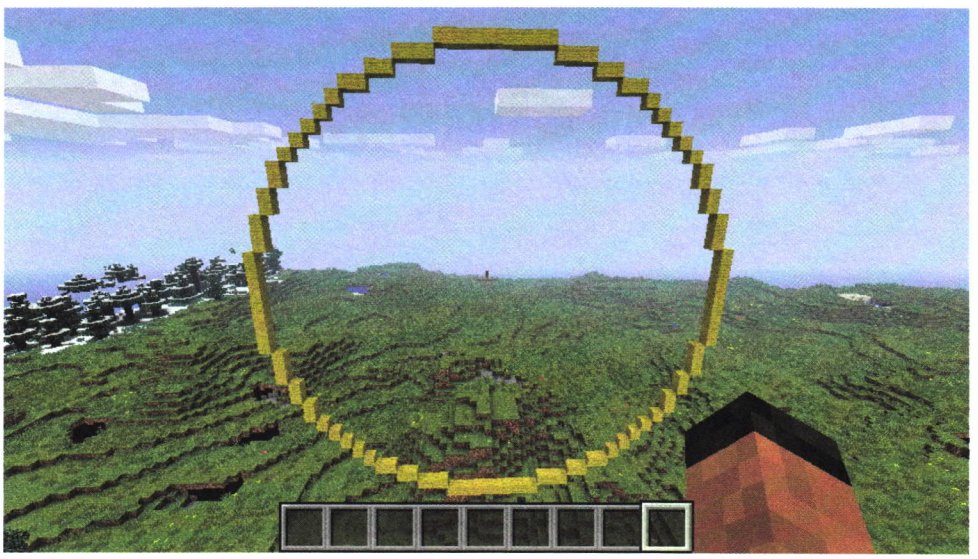


Рис. 6.5. Определение точки на циферблате, куда указывает стрелка

Тригонометрия — раздел математики, изучающий отношения между углами и длинами сторон треугольника. Посетите страницу https://ru.wikipedia.org/wiki/Тригонометрия, чтобы узнать больше.

Чтобы построить часы, выполните следующие шаги:

1. Создайте новую программу для Minecraft в редакторе IDLE и сохраните ее в папке *MyAdventures* с именем *MinecraftClock.py*.

2. Импортируйте модули `minecraft`, `block`, `minecraftstuff`, `time`, `datetime` и `math`:

```
import mcpi.minecraft as Minecraft
import mcpi.block as block
import mcpi.minecraftstuff as minecraftstuff
import time
import datetime
import math
```

3. Создайте функцию `findPointOnCircle()`. Она должна принимать координаты центра окружности и угол поворота стрелки часов, а также возвращать координаты свободного конца стрелки, как показано на рис. 6.5.

```
def findPointOnCircle(cx, cy, radius, angle):
  x = cx + math.sin(math.radians(angle)) * radius
  y = cy + math.cos(math.radians(angle)) * radius
  x = int(round(x, 0))
  y = int(round(y, 0))
  return(x,y)
```

4. Подключитесь к игре Minecraft и создайте объект `MinecraftDrawing`:

```
mc = minecraft.Minecraft.create()
mcdrawing = minecraftstuff.MinecraftDrawing(mc)
```

5. Определите текущую позицию персонажа:

```
pos = mc.player.getTilePos()
```

6. Теперь создайте переменные для хранения координат центра циферблата часов (будет находиться на высоте 25 блоков над персонажем), радиуса циферблата и длины стрелок:

```
clockMiddle = pos
clockMiddle.y = clockMiddle.y + 25
CLOCK_RADIUS = 20
HOUR_HAND_LENGTH = 10
MIN_HAND_LENGTH = 18
SEC_HAND_LENGTH = 20
```

7. Далее постройте циферблат с помощью `drawCircle()`:

```
mcdrawing.drawCircle(
    clockMiddle.x, clockMiddle.y, clockMiddle.z,
    CLOCK_RADIUS, block.DIAMOND_BLOCK.id)
```

8. Программа еще не завершена, но вы должны запустить ее сейчас, чтобы проверить, все ли работает, и появляется ли над персонажем большой круг (циферблат).

Далее нужно добавить код в программу для того, чтобы на часах появились стрелки.

1. Начните бесконечный цикл. Весь код, следующий за этой строкой, будет принадлежать циклу:

```
while True:
```

2. Затем надо посмотреть на компьютере текущее время, вызвав функцию `datetime.datetime.now()`, и разбить его на часы, минуты, секунды. Так как на циферблате есть

всего 12 часов, а не 24, из времени после полудня нужно вычесть 12 часов (например, если текущее время равно 14:00, вычитаем 12, чтобы получить 2 часа). Для этого введите следующий код:

```
timeNow = datetime.datetime.now()
hours = timeNow.hour
if hours >= 12:
  hours = timeNow.hour - 12
minutes = timeNow.minute
seconds = timeNow.second
```

3. Постройте часовую стрелку. Угол поворота стрелки можно получить, разделив 360 градусов на 12 часов и умножив результат на текущее количество часов. Найдите координаты x и y свободного конца стрелки с помощью `findPointOnCircle()` и постройте ее вызовом `drawLine()`:

```
hourHandAngle = (360 / 12) * hours

hourHandX, hourHandY = findPointOnCircle(
    clockMiddle.x, clockMiddle.y,
    HOUR_HAND_LENGTH, hourHandAngle)

mcdrawing.drawLine(
    clockMiddle.x, clockMiddle.y, clockMiddle.z,
    hourHandX, hourHandY, clockMiddle.z,
    block.DIRT.id)
```

4. Далее постройте минутную стрелку, расположив ее на один блок дальше часовой стрелки (`z-1`):

```
minHandAngle = (360 / 60) * minutes

minHandX, minHandY = findPointOnCircle(
    clockMiddle.x, clockMiddle.y,
    MIN_HAND_LENGTH, minHandAngle)

mcdrawing.drawLine(
    clockMiddle.x, clockMiddle.y, clockMiddle.z-1,
    minHandX, minHandY, clockMiddle.z-1,
    block.WOOD_PLANKS.id)
```

5. Добавьте секундную стрелку, расположив ее на один блок ближе часовой стрелки (`z+1`):

```
secHandAngle = (360 / 60) * seconds

secHandX, secHandY = findPointOnCircle(
    clockMiddle.x, clockMiddle.y,
    SEC_HAND_LENGTH, secHandAngle)

mcdrawing.drawLine(
    clockMiddle.x, clockMiddle.y, clockMiddle.z+1,
    secHandX, secHandY, clockMiddle.z+1,
    block.STONE.id)
```

6. Добавьте задержку в 1 секунду:

```
time.sleep(1)
```

7. Программа пока еще не закончена, но вы должны запустить ее, чтобы проверить, что появились стрелки. Посмотрите, что происходит, когда стрелки двигаются.

   Возможно, вы заметили, что стрелки часов не стираются, когда двигаются. Следующий шаг — стереть стрелки часов, а затем нарисовать их снова с помощью `AIR`.

8. Сотрите часовую стрелку:

```
mcdrawing.drawLine(
    clockMiddle.x, clockMiddle.y, clockMiddle.z,
    hourHandX, hourHandY, clockMiddle.z,
    block.AIR.id)
```

9. Сотрите минутную стрелку:

```
mcdrawing.drawLine(
    clockMiddle.x, clockMiddle.y, clockMiddle.z-1,
    minHandX, minHandY, clockMiddle.z-1,
    block.AIR.id)
```

10. Сотрите секундную стрелку:

```
mcdrawing.drawLine(
    clockMiddle.x, clockMiddle.y, clockMiddle.z+1,
    secHandX, secHandY, clockMiddle.z+1,
    block.AIR.id)
```

11. Сохраните и запустите программу, чтобы увидеть плоды своего труда: гигантские часы над игроком. Но имейте в виду: чтобы узнать время, надо смотреть на циферблат с лицевой стороны, в противном случае возникнет впечатление, что время идет назад!

Полный программный код, создающий часы, можно найти на веб-сайте книги www.wiley.com/go/adventuresinminecraft2e, но я настоятельно рекомендую ввести весь код вручную, по ходу чтения инструкций. Так вы узнаете намного больше!

УГЛУБЛЯЕМСЯ В КОД

Функция `findPointOnCircle()` вычисляет координаты точки (x, y) на окружности, исходя из координат центра окружности (cx, cy), радиуса и угла (см. рис. 6.5).

1. Функция принимает следующие входные параметры: координаты центра окружности (`cx`, `cy`), радиус (`radius`) и угол (`angle`):

```
def findPointOnCircle(cx, cy, radius, angle):
```

2. Координаты точки на окружности (x, y) вычисляются умножением результатов функций `sin()` и `cos()` из модуля `math` на радиус:

```
  x = cx + math.sin(math.radians(angle)) * radius
  y = cy + math.cos(math.radians(angle)) * radius
```

Функции `math.sin()` и `math.cos()` принимают величину угла в **радианах**. Радианы — еще одна единица измерения углов (вместо часто используемых градусов). Для преобразования величин углов в радианы используется функция `math.radians()`.

3. Значения `x` и `y` вычисляются как вещественные числа (с дробной частью), но функция должна вернуть целые значения, поэтому для округления вещественных чисел и преобразования их в целые числа используют функции `round()` и `int()`:

```
x = int(round(x, 0))
y = int(round(y, 0))
```

4. Далее значения `x` и `y` возвращаются программе:

```
return(x, y)
```

Функция `findPointOnCircle()` возвращает сразу обе координаты (x, y), поэтому вызывающая программа должна предоставить две переменные при вызове функции:

```
x, y = findPointOnCircle(cx, cy, radius, angle)
```

Стрелки часов создаются в три этапа:

1. Вычислить угол стрелки, разделив 360 градусов на 12 или на 60 (в зависимости от типа стрелки — часовая, минутная или секундная), и умножить полученное частное на число часов, минут или секунд:

```
hourHandAngle = (360 / 12) * hours
```

2. Найти координаты (x, y) свободного конца стрелки с помощью функции `findPointOnCircle()`:

```
hourHandX, hourHandY = findPointOnCircle( ↩
   clockMiddle.x, clockMiddle.y,
   HOUR_HAND_LENGTH, hourHandAngle)
```

Так как функция возвращает сразу два значения, при ее вызове надо указать две переменные (`hourHandX` , `hourHandY`).

3. Построить линию из центра циферблата к найденной точке на окружности:

```
mcdrawing.drawLine( ↩
    clockMiddle.x, clockMiddle.y, clockMiddle.z,
    hourHandX, hourHandY, clockMiddle.z,
    block.DIRT.id)
```

Позиция часовой стрелки на настоящих часах заметно меняется с каждой минутой. Например, в 11:30 часовая стрелка указывает на середину между 11 и 12. Программа, которую вы только что написали, работает иначе: часовая стрелка продолжит указывать на 11 вплоть до момента, когда время 11:59:59 изменится на 12:00:00.

Подумайте над этим. Возможно, вам удастся самостоятельно изменить программу — так, чтобы часы в мире Minecraft работали подобно настоящим часам, меняя угол поворота часовой стрелки в соответствии с текущим временем.

Создание многоугольников

Многоугольник — это плоская двумерная фигура, образованная несколькими соединенными между собой отрезками. Многоугольники могут иметь любое количество сторон — от трех (треугольник) и более. Как показано на рис. 6.6, в мире Minecraft легко создать множество интересных многоугольников.

Рис. 6.6. Примеры многоугольников в Minecraft

Даже при том, что это двумерные фигуры, их с успехом можно использовать в трехмерной графике для создания практически любых трехмерных объектов, состоящих из множества многоугольников. Когда трехмерные объекты конструируются из многоугольников, последние называют **гранями**. Так можно создать массу интересных конструкций. Взгляните на рис. 6.7, где изображена панорама острова Манхэттен, созданная из множества многоугольников (увидеть исходный код программы, создавшей эту панораму, можно по адресу: www.stuffaboutcode.com/2013/04/minecraft-pi-edition-manhattan-stroll.html).

Грань — это одна плоская поверхность, являющаяся частью большого объекта, например одна сторона куба или верхняя крышка бочки.

Рис. 6.7. Панорама острова Манхэттен в Нью-Йорке

Создавать многоугольники (грани) можно с помощью функции `drawFace()` объекта `MinecraftDrawing`. Функция принимает список точек (*x*, *y*, *z*), которые последовательно соединены отрезками для формирования замкнутого многоугольника. Передача значения `True` или `False` определяет необходимость создания сплошной грани; последний параметр указывает тип блоков для строительства грани (рис. 6.8):

```
drawFace(points, filled, blockType, blockData)
```

Создайте новую программу для экспериментов с функцией `drawFace()` и постройте треугольник, как показано на рис. 6.8:

1. Откройте редактор IDLE и создайте новый файл. Сохраните его в папке *MyAdventures* с именем *Polygon.py*.
2. Импортируйте модули `minecraft`, `block` и `minecraftstuff`:

```
import mcpi.minecraft as minecraft
import mcpi.block as block
import mcpi.minecraftstuff as minecraftstuff
```

3. Подключитесь к игре Minecraft и создайте объект `MinecraftDrawing`:

```
mc = minecraft.Minecraft.create()
mcdrawing = minecraftstuff.MinecraftDrawing(mc)
```

4. Определите координаты персонажа:

```
pos = mc.player.getTilePos()
```

5. С помощью класса `minecraftstuff.Points` создайте список точек:

```
points = minecraftstuff.Points()
```

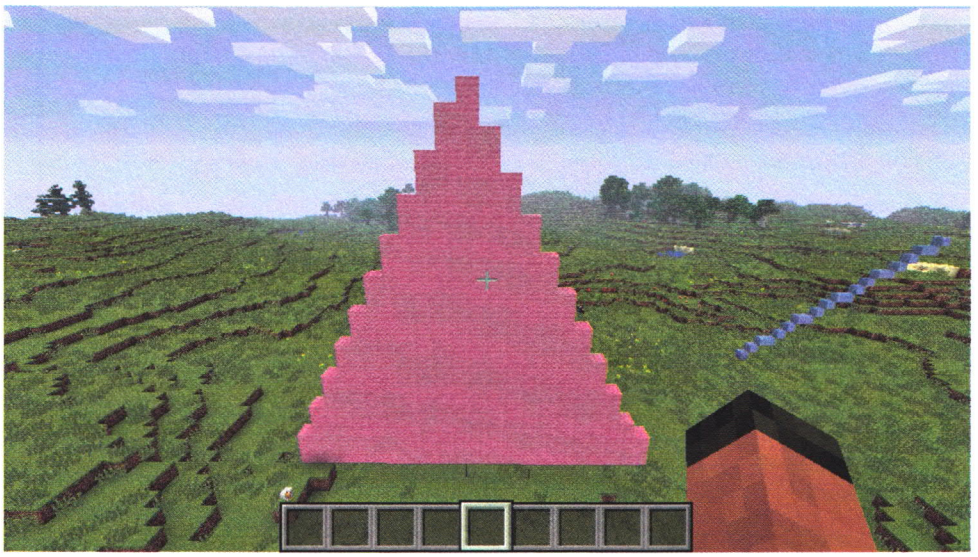


Рис. 6.8. Использование drawFace() для создания треугольной формы из трех точек

6. Добавьте в список `points` три точки (x, y, z), которые будут соединены для образования треугольника:

```
points.add(pos.x, pos.y, pos.z)
points.add(pos.x + 20, pos.y, pos.z)
points.add(pos.x + 10, pos.y + 20, pos.z)
```

7. С помощью функции `MinecraftDrawing.drawFace` постройте треугольник:

```
mcdrawing.drawFace(points, True, block.WOOL.id, 6)
```

8. Сохраните и запустите программу, чтобы построить треугольник.

Какие еще фигуры может строить функция `drawFace()`? Можно ли с ее помощью построить пятиугольник, например пентагон?

Пирамиды

Найдите изображение пирамиды и внимательно его рассмотрите. Что вы заметили? Какую форму имеют грани пирамиды? Что у них общего? Сколько всего граней?

Как вы уже, наверное, знаете, каждая грань пирамиды (кроме основания) является треугольником. Египетские пирамиды имеют четыре стороны (или пять, если считать основа-

ние), но в принципе они могут иметь любое количество сторон — от трех и более. Также обратите внимание на то, что основание пирамиды всегда вписывается в окружность! Взгляните на рис. 6.9, чтобы понять, о чем я говорю.

Теперь вы напишете программу, которая с помощью функций `drawFace()` и `findPointOnCircle()` будет строить пирамиды любого размера, любой высоты и с любым числом граней:

1. Откройте редактор IDLE и создайте новый файл. Сохраните его в папке *MyAdventures* с именем *MinecraftPyramids.py*.
2. Импортируйте модули `minecraft`, `block`, `minecraftstuff` и `math`:

```
import mcpi.minecraft as minecraft
import mcpi.block as block
import mcpi.minecraftstuff as minecraftstuff
import math
```

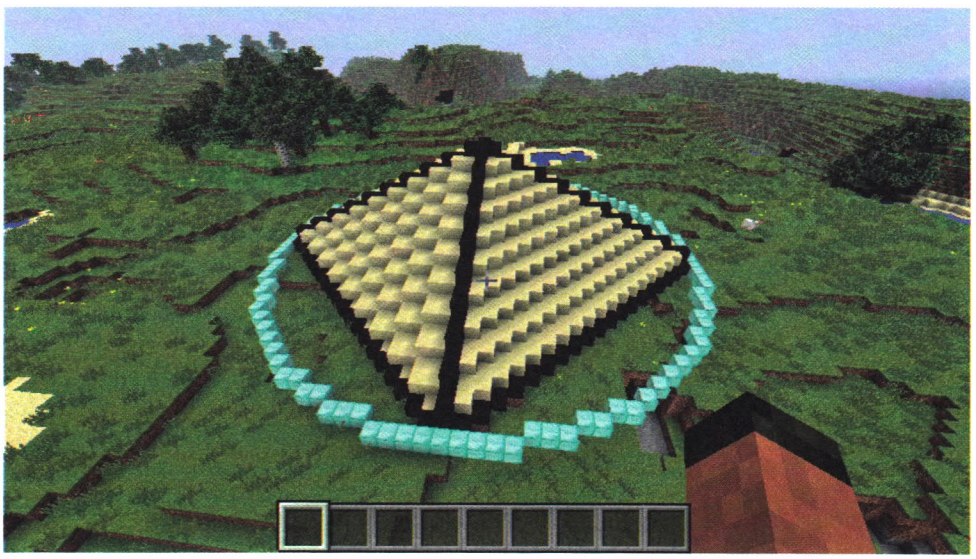


Рис. 6.9. Пирамида, построенная из треугольников, с основанием, точно вписывающимся в окружность

3. Создайте функцию `findPointOnCircle()`, которая будет вычислять точку на окружности, где должен быть помещен каждый треугольник:

```
def findPointOnCircle(cx, cy, radius, angle):
  x = cx + math.sin(math.radians(angle)) * radius
  y = cy + math.cos(math.radians(angle)) * radius
  x = int(round(x, 0))
  y = int(round(y, 0))
  return(x,y)
```

4. Подключитесь к миру Minecraft и создайте объект `MinecraftDrawing`:

```
mc = minecraft.Minecraft.create()
mcdrawing = minecraftstuff.MinecraftDrawing(mc)
```

5. Подготовьте переменные, необходимые для строительства пирамиды. Центр основания пирамиды будет находиться в позиции персонажа. Значения этих переменных определяют размер (радиус), высоту и число граней пирамиды:

```
pyramidMiddle = mc.player.getTilePos()
RADIUS = 20
HEIGHT = 10
SIDES = 4
```

6. Выполните цикл по числу граней пирамиды; весь программный код, следующий ниже этой строки, принадлежит циклу `for`:

```
for side in range(0, SIDES):
```

Чем больше пирамида, тем дольше программа будет ее строить, а игра Minecraft — отображать. Если пирамида слишком высокая, она может превысить максимальную высоту мира в игре. Поэтому не спешите и увеличивайте значения постепенно. Вы можете создавать огромные пирамиды, но для этого нужно терпение, так как для отображения по-настоящему больших конструкций игре требуется время.

7. Для каждой грани пирамиды вычислите углы, затем с помощью `findPointOnCircle()` найдите координаты *x*, *y*, *z*. Угол вычисляется делением 360 градусов на общее число сторон и умножением на порядковый номер текущей грани, как показано на рис. 6.10. Введите следующий код с отступами в цикле `for`:

```
  point1Angle = int(round((360 / SIDES) * side,0))
  point1X, point1Z = findPointOnCircle(
    middle.x, middle.z, RADIUS, point1Angle)

point2Angle = int(round((360 / SIDES) * (side + 1),0))
point2X, point2Z = findPointOnCircle(
     middle.x, middle.z, RADIUS, point2Angle)
```

8. Определите координаты вершин треугольника и с помощью `drawFace()` постройте грань пирамиды:

```
points = minecraftstuff.Points()
points.add(point1X, middle.y, point1Z)
points.add(point2X, middle.y, point2Z)
points.add(middle.x, middle.y + HEIGHT, middle.z)
mcdrawing.drawFace(points, True, block.SANDSTONE.id)
```

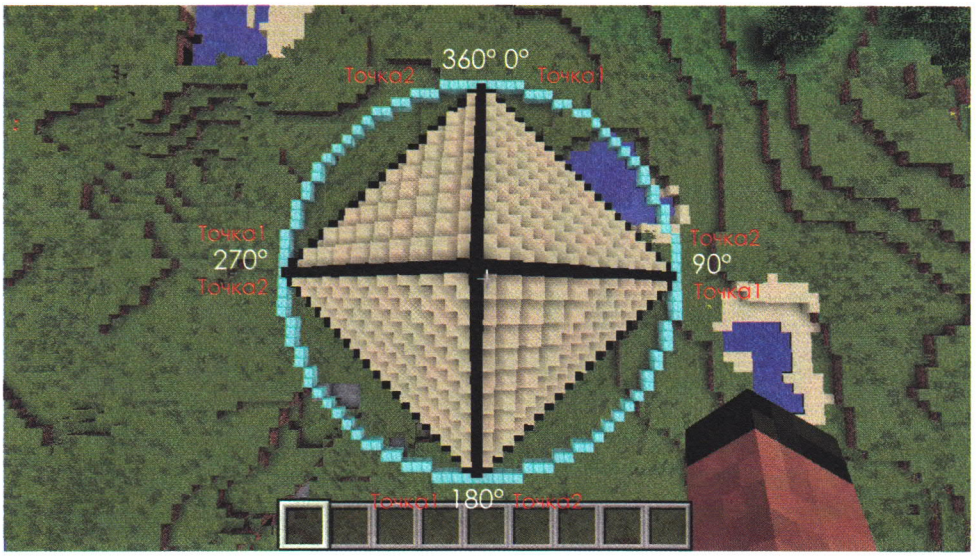


Рис. 6.10. Углы для пирамиды с четырьмя гранями

Песчаник (`block.SANDSTONE.id`) замечательно подходит для строительства пирамид, поскольку напоминает песок, но имеет более подходящие для строительства характеристики: не осыпается и падает вниз, если выбить нижний блок из-под верхнего. Если построить пирамиду из песка, персонаж будет завален песком и ему придется потратить много сил, чтобы выкопать себя.

9. Сохраните и запустите программу. Вы увидите, как пирамида будет построена вокруг персонажа и поймает его словно в ловушку!

Эта программа может строить пирамиды любого размера и с любым количеством сторон. Попробуйте изменить в программе переменные, определяющие параметры пирамиды, и запустите ее повторно. На рис. 6.11 дана пара интересных примеров.

Полный программный код строительства пирамид есть на веб-сайте книги www.wiley.com/go/adventuresinminecraft2e.

Пирамида, построенная вами, не имеет основания. Сможете ли вы сами создать для него многоугольник? Для четырехгранной пирамиды это легко сделать: если вы не допустите ошибок, тот же код сумеет построить и пяти-, и шести-, и даже семигранные пирамиды.

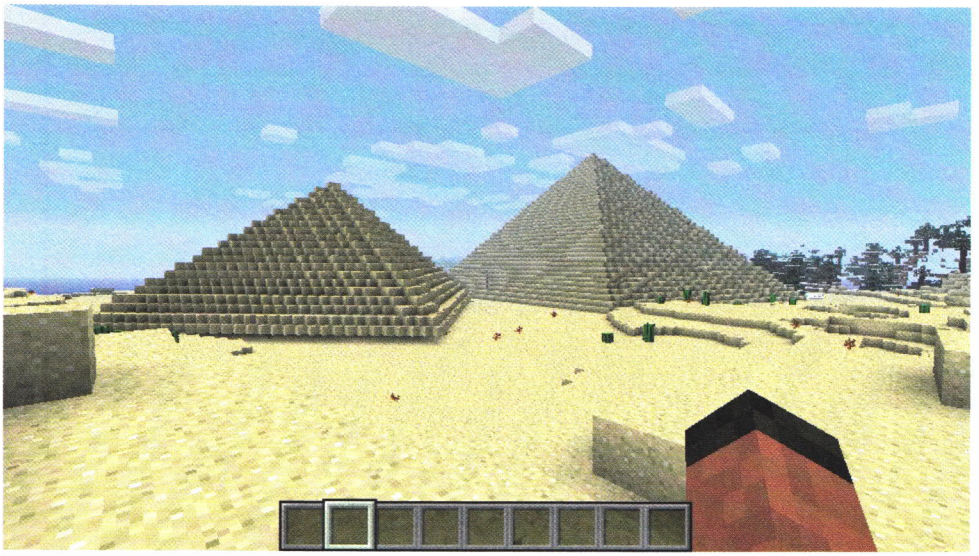


Рис. 6.11. Пирамиды в Minecraft

Дополнительные приключения с двух- и трехмерными фигурами

С помощью функции `drawFace()` можно создать любой **многогранник**, образованный плоскими гранями (как пирамиды, которые вы создавали выше). Почему бы не попробовать создать их?

Множество примеров многогранников и идей вы найдете на сайтах:

- Математические этюды (http://www.etudes.ru/ru/etudes/platonic/).
- Журнал «Волшебные грани» (http://mnogogranniki.ru/vidy-mnogogrannikov.html).

Краткая справочная таблица

Команды	Описание
`mcdrawing.drawLine(0, 0, 0, 10, 10, 10, block.DIRT.id)`	Строит линию, соединяющую две точки
`mcdrawing.drawCircle(0, 0, 0, radius, block.DIRT.id)`	Строит окружность

Краткая справочная таблица

Команды	Описание

`mcdrawing.drawSphere(0, 0, 0,`
`radius,`
`block.DIRT.id)`
	Строит сферу

`tri = []`
`filled = True`
`tri.add(0,0,0))`
`tri.add(10,0,0))`
`tri. add (5,10,0))`
`mcdrawing.drawFace(tri,`
`filled,`
`block.DIRT.id)`
	Строит многоугольник или грань (например, треугольник)

Новое достижение: мастер строительства трехмерных объектов и создатель огромных конструкций. Здравствуй, строитель пирамид!

В СЛЕДУЮЩЕМ ПРИКЛЮЧЕНИИ

В следующем Приключении вы узнаете, как наделить объекты мира Minecraft разумом, подружиться с блоком и избежать вторжения инопланетян.

Приключение 7
Наделение блоков способностью мыслить

КОМПЬЮТЕРЫ НЕ ДУМАЮТ. Они вообще не способны мыслить и делают лишь то, что скажет программист. Однако вы можете придать компьютеру вид мыслящей и принимающей решения конструкции, запрограммировав «понимание» происходящего и снабдив компьютер правилами выбора «решения», что делать дальше. Тем самым вы откроете для себя массу интересного.

В этом Приключении вы запрограммируете дружественный блок — блокфренд — который будет следовать за вами повсюду, не выпуская вас из виду. Также вы создадите летающую тарелку, преследующую персонажа, которая, оказавшись над ним, попытается телепортировать его внутрь.

Еще вы узнаете, как заставить блок двигаться и выбирать самый короткий путь, а также как с помощью модуля `random` придать компьютеру вид мыслящего существа. Кроме того, вы создадите фигуры с помощью функций `MinecraftShape` из модуля `minecraftstuff` (включен в начальный комплект инструментов).

Ваш блокфренд

Minecraft может быть пустынным миром. Но ваш персонаж не обязан томиться в одиночестве: вы можете создать блок, повсюду следующий за ним, который будет разговаривать и станет его другом (рис. 7.1).

Чтобы запрограммировать блок, ставший другом персонажу, вам нужно научиться думать, как он. Он будет счастлив находиться рядом с персонажем, всюду ходить за ним и стараться быть как можно ближе. Если персонаж отойдет, блокфренд последует за ним. Если персонаж уйдет слишком далеко, блок загрустит, прекратит движение и будет стоять на месте, пока персонаж не вернется и не обнимет его (встанет рядом с блоком).

Программа блокфренда делится на две части:

- **Правила, с помощью которых блок решает, что делать дальше**: эта часть программы поможет блоку решить, стоит ли продолжать идти за персонажем (целью) или лучше остановиться.
- **Программный код, перемещающий блок за персонажем**: достигнув цели, блок снова будет опираться на правила, желая определить, что делать дальше.

Рис. 7.1. Создайте блокфренда, который будет сопровождать вашего персонажа в приключениях

Следуя за целью, он должен перемещаться по поверхности земли, не летать по воздуху и не прокладывать себе путь под землей! Добиться этого можно, вызвав функцию `mc.getHeight(x,z)` и передав ей горизонтальные координаты (x, z). В ответ функция вернет вертикальную координату (y) первого сверху блока, тип которого отличается от `AIR`.

На сайте книги www.wiley.com/go/adventuresinminecraft2e есть видеоурок, где показано, как создать блокфренда.

Напишите новую программу для создания блокфренда:

1. Откройте редактор IDLE, создайте новый файл, выбрав в меню пункт File ▸ New File (Файл ▸ Создать файл), и сохраните его в папке *MyAdventures* с именем *BlockFriend.py*.
2. Импортируйте модули `minecraft`, `block`, `minecraftstuff`, `math` и `time`:

```
import mcpi.minecraft as minecraft
import mcpi.block as block
import mcpi.minecraftstuff as minecraftstuff
import math
import time
```

3. Первое, что нужно сделать, — создать функцию для вычисления расстояния между двумя точками. С ее помощью блок определит, как далеко он находится от персонажа:

```
def distanceBetweenPoints(point1, point2):
  xd = point2.x - point1.x
  yd = point2.y - point1.y
  zd = point2.z - point1.z
  return math.sqrt((xd*xd) + (yd*yd) + (zd*zd))
```

4. Теперь надо решить, как далеко должен оказаться персонаж, чтобы блок перестал за ним следовать. Создайте константу для хранения расстояния, которое «слишком далеко». Я выбрал число 15, то есть когда блок и персонаж окажутся на расстоянии 15 блоков друг от друга, блокфренд прекратит следовать за персонажем:

```
TOO_FAR_AWAY = 15
```

5. Создайте объекты `Minecraft` и `MinecraftDrawing`:

```
mc = minecraft.Minecraft.create()
mcdrawing = minecraftstuff.MinecraftDrawing(mc)
```

6. Создайте переменную для хранения настроения блока. В этом Приключении блок счастлив (happy) либо расстроен (sad). Присвойте переменной значение `"happy"` (счастлив):

```
blockMood = "happy"
```

7. Создайте блок на расстоянии нескольких блоков от персонажа, для чего сначала определите его позицию, затем добавьте 5 к координате *x* и с помощью функции `getHeight()` — координату *y*, чтобы блок оказался на поверхности земли:

```
friend = mc.player.getTilePos()
friend.x = friend.x + 5
friend.y = mc.getHeight(friend.x, friend.z)
mc.setBlock(friend.x, friend.y, friend.z,
            block.DIAMOND_BLOCK.id)
mc.postToChat("<block> Hello friend")
```

8. Создайте цель для блока, к которой он будет стремиться. Первоначально определите координаты цели, совпадающие с текущими координатами блока, чтобы тот не начал движение сразу после запуска программы:

```
target = friend.clone()
```

Чтобы скопировать координаты в мире Minecraft, нужно использовать функцию `clone()`: функции Minecraft API возвращают координаты в виде объектов языка Python, отличающихся от обычных переменных. Например, если выполнить копирование инструкцией `target = friend` и потом изменить значение `friend.x` в объекте `friend`, значение `target.x` в объекте `target` тоже изменится.

9. Начните бесконечный цикл, чтобы программа выполняла его вечно. (Обратите внимание: весь программный код ниже принадлежит этому циклу.)

```
while True:
```

10. Получите координаты персонажа и вычислите расстояние между персонажем и блокфренд с помощью функции `distanceBetweenPoints()`:

```
pos = mc.player.getTilePos()
distance = distanceBetweenPoints(pos, friend)
```

11. Примените правила, чтобы блокфренд определил, что делать дальше. Если он «счастлив» (happy), предложите ему сравнить расстояние до персонажа с константой «слишком далеко». Если расстояние меньше, чем «слишком далеко», переместите цель в позицию персонажа, а если больше — измените настроение на «расстроен» (sad) (рис. 7.2):

```
if blockMood == "happy":
  if distance < TOO_FAR_AWAY:
    target = pos.clone()
  else:
    blockMood = "sad"
    mc.postToChat("<block> Come back. You are too far ↩
                   away. I need a hug!")
```

12. Теперь надо указать, что в противном случае (когда расстояние не меньше, чем «слишком далеко»), блок «расстроен» и останавливается, чтобы дождаться, пока персонаж подойдет к нему на расстояние одного блока (расстояние объятий), после чего настроение блока изменится на «счастлив»:

```
elif blockMood == "sad":
  if distance <= 1:
    blockMood = "happy"
    mc.postToChat("<block> Awww thanks. Let's go.")
```

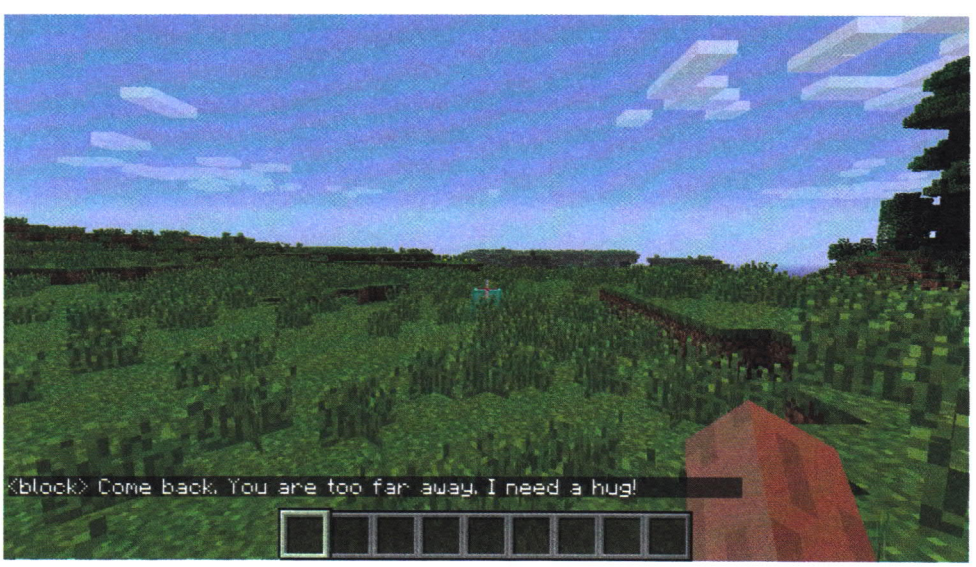


Рис. 7.2. Блокфренд расстроен

13. Запустите программу, чтобы проверить, как меняется настроение вашего блокфренда — от счастливого до грустного, когда вы уходите, — и как он снова становится счастливым, когда вы обнимаете его.

    Блокфренд появляется рядом с персонажем. Если вы уходите слишком далеко, то друг просит вас вернуться.

    Теперь, когда ваш блокфренд разговаривает с вами, чтобы вы знали, как он себя чувствует, следующим шагом будет обновление программы, чтобы заставить блокфренда следовать за персонажем:

1. Блокфренду нужно лишь двигаться, если его позиция не совпадает с его целью (персонажем):

```
if friend != target:
```

2. Найдите все блоки между другом и его целью с помощью функции `getLine()` объекта `MinecraftDrawing`:

```
line = mcdrawing.getLine(
  friend.x, friend.y, friend.z,
  target.x, target.y, target.z)
```

    Функция `getLine()` действует примерно так же, как функция `drawLine()` (см. Приключение 6). Только вместо строительства линии из блоков она возвращает список координат (*x*, *y*, *z*) блоков, находящихся на прямой линии между двумя указанными заданными множествами координат *x*, *y*, *z*.

3. Непосредственно под предыдущим кодом добавьте цикл, который выполнит итерации по всем блокам между другом и целью и переместит друга к цели:

```
for blockBetween in blocksBetween[:-1]:
  mc.setBlock(friend.x, friend.y, friend.z, block.AIR.id)
  friend = blockBetween.clone()
  friend.y = mc.getHeight(friend.x, friend.z)
  mc.setBlock(friend.x, friend.y, friend.z,
              block.DIAMOND_BLOCK.id)
  time.sleep(0.25)
target = friend.clone()
```

Когда цикл `for` завершится, блокфренд достигнет своей цели. Поэтому скопируйте координаты блока в координаты цели. В данном случае блок прекратит движение.

Программа перемещает блок, очищая его в предыдущей позиции (это объясняет, почему устанавливается тип блока `AIR`), меняя координаты блокфренда на координаты следующего блока в линии и воссоздавая блок в этой позиции.

Скорость движения блока определяется задержкой после перемещения друга к следующему блоку — `time.sleep(0.25)`. Если задержка слишком короткая, блок будет двигаться слишком быстро и персонаж не сможет от него убежать, если чересчур долгая — блок будет двигаться медленно и раздражать вас своей медлительностью.

4. Добавьте небольшую задержку в конце бесконечного цикла, чтобы программа запрашивала позицию персонажа каждые 0,25 секунды:

```
time.sleep(0.25)
```

5. Запустите программу.

Блокфренд появится недалеко от персонажа и будет следовать за ним, пока они не окажутся слишком далеко друг от друга. Когда это случится, блокфренд остановится и персонажу придется подойти к нему, встать рядом, прежде чем тот продолжит «преследование».

Полный код программы можно загрузить на сайте книги www.wiley.com/go/adventuresinminecraft2e.

УГЛУБЛЯЕМСЯ В КОД

Для вычисления расстояния между блокфрендом и персонажем вы написали функцию с именем `distanceBetweenPoints(point1, point2)`. Вычисление точного расстояния между двумя точками производится на основе теоремы Пифагора. Если помните, в Приключении 4 вам уже предлагалось подумать, как применить теорему Пифагора, в разделе «Добавление вывода подсказок».

Вычисление расстояния выполняется, как описано ниже:

1. Находится разность между значениями *x*, *y*, *z* в `point1` и `point2`:

```
xd = point2.x - point1.x
yd = point2.y - point1.y
zd = point2.z - point1.z
```

2. Вычисляются квадраты разностей координат точек:

```
xd*xd
```

3. Вычисляется сумма квадратов разностей:

```
(xd*xd) + (yd*yd) + (zd*zd)
```

4. Вызывающей программе возвращается расстояние между двумя точками, которое вычисляется как **квадратный корень** из суммы квадратов разностей с помощью функции `math.sqrt()`:

```
return math.sqrt((xd*xd) + (yd*yd) + (zd*zd))
```

Посетите страницу *www.mathsisfun.com/algebra/distance-2-points.html*, чтобы познакомиться с теоремой Пифагора и узнать, как ее можно использовать, чтобы найти расстояние между двумя точками.

Блокфренд перемещается в сторону персонажа, отыскивая блоки между ним и персонажем, а затем в цикле двигается по этим блокам:

```
for blockBetween in blocksBetween[:-1]:
```

Здесь `[:-1]` сообщает интерпретатору Python, что цикл должен обойти все блоки в списке `line`, пока не окажется на блоке рядом с персонажем, а не у него на голове.

Квадратный корень числа — значение, которое можно умножить само на себя и получить это число. Например, корень квадратный из 9 равен 3, потому что 3 × 3 = 9.

Друг всегда перемещается с постоянной скоростью: на один блок — 0,25 секунды, или четыре блока в секунду. Однако в жизни друзья обычно шагают тем быстрее, чем дальше находятся от цели. Попробуйте изменить программу, чтобы скорость движения блока увеличивалась с расстоянием между ним и персонажем.

Вернитесь к разделу «Добавление вывода подсказок» в Приключении 4 и выполните предложенное там задание, но на сей раз используйте функцию `distanceBetweenPoints()`, чтобы получить лучшую оценку расстояния.

Использование случайных чисел с целью разнообразить поведение друга

Проблема с импровизированным другом в том, что он слишком **предсказуем**, всегда делает одно и то же: запустив программу пару раз, вы будете точно знать, что и когда он делает. Поэтому игра очень быстро надоедает. Чтобы придать блокфренду подобие «ума», нужно добавить в его поведение элемент непредсказуемости.

Когда что-то **предсказуемо**, это значит, что вы можете предвидеть, что случится еще до того, как оно произойдет. Несложно добавить случайности и получить поведение, близкое к реальности. Ведь события в реальной жизни далеко не всегда можно предсказать.

С помощью случайных чисел можно сымитировать непредсказуемость — сделать что-то менее предсказуемым. Для этого нужно заставить программу делать выбор между решениями, основываясь на **вероятности**. Например, выполнять какое-то действие в одном случае из ста. Добавив больше правил с разными вероятностями, можно сделать программу почти непредсказуемой.

Прежде чем вносить в программу блокфренда изменения, добавляющие элемент случайности, исследуем программный код создания случайных чисел и проверки вероятности. Возможно, вы помните, что познакомились со случайными числами в Приключении 3.

Вероятность — это мера возможности некоторого события. Например, если подбросить монету вверх, есть вероятность 50 % (один шанс из двух), что она упадет решкой вверх.

Модуль `random` в языке Python содержит функцию `random.randint(startNumber, endNumber)`, генерирующую случайные числа в диапазоне между указанными числами (`startNumber`, `endNumber`).

Следующая программа выводит случайное число в диапазоне от 1 до 10 при каждом запуске. Чтобы убедиться в этом, создайте новую программу:

```
import random
randomNo = random.randint(1,10)
print randomNo
```

Добавив инструкцию `if`, сравнивающую случайное число с 10, можно создать проверку вероятности, которая будет заканчиваться успехом примерно один раз из десяти случаев:

```
import random
if random.randint(1,10) == 10:
  print "This happens about 1 time in 10"
else
  print "This happens about 9 times out of 10"
```

Если запустить эту программу 100 раз, можно ожидать, что сообщение «This happens about 1 time in 10» («это происходит в одном случае из десяти») появится примерно 10 раз (рис. 7.3). Но точно так же оно может появиться девять или 11 раз или вообще не появиться. Вот она, непредсказуемость!

Как вы думаете, если запустить эту программу миллион раз, сколько раз будет выведено сообщение «This happens about 1 time in 10» («Это происходит в одном случае из десяти»)? Можно ожидать, что оно появится 100 000 раз, но ведь может вообще не появиться или может появиться миллион раз!

Если использовать случайные числа и проверку вероятности в программе, создающей друга, можно сделать его поведение менее предсказуемым или даже создать блок, являющийся недругом персонажа.

Добавьте несколько новых правил в программу, чтобы в одном случае из ста «расстроенный» блокфренд мог решить, что он ждал слишком долго и больше не будет следовать за персонажем, когда тот вернется и обнимет его:

1. Откройте редактор IDLE, а в нем — программу *BlockFriend.py* из папки *MyAdventures*.
2. Выберите пункт меню File ▸ Save As (Файл ▸ Сохранить как) и сохраните файл с именем *BlockFriendRandom.py*.

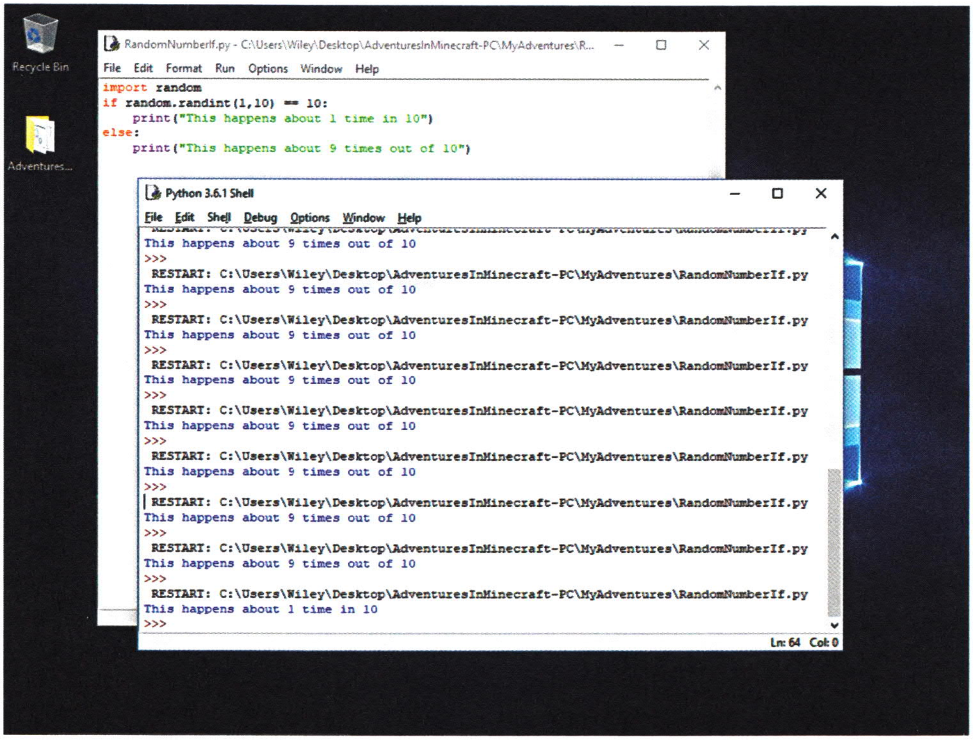


Рис. 7.3. Создание случайных чисел, с помощью которых можно сделать друга менее предсказуемым

3. Добавьте в начало программы инструкцию `import`, чтобы импортировать модуль `random` (выделена **жирным шрифтом**):

```
import mcpi.minecraft as minecraft
import mcpi.block as block
import mcpi.minecraftstuff as minecraftstuff
import math
import time
import random
```

4. Добавьте проверку одного случая из ста, чтобы изменить настроение блока на `"hadenough"` (с меня хватит), если он находится в настроении `"sad"` (расстроен):

```
  elif blockMood == "sad":
    if distance <= 1:
      blockMood = "happy"
      mc.postToChat("<block> Awww thanks. Let's go.")
    if random.randint(1,100) == 100:
      blockMood = "hadenough"
      mc.postToChat(" <block> That's it. I have had ↩
       enough.»)
```

5. Сохраните и запустите программу.

Если увести персонаж слишком далеко от блокфренда и подождать достаточно долго (чтобы выпал один шанс из ста), блокфренд решит, что ждал слишком долго, и перестанет быть другом. Когда это произойдет, он скажет: «That's it. I have had enough» («Вот и все. С меня довольно») — и навсегда останется на своем месте.

Измените программу так, чтобы в одном случае из пятидесяти «обиженный» друг забывал свою обиду на персонажа, когда тот к нему подойдет.

Большие фигуры

В предыдущей программе вы создали друга в виде единственного блока, который следует за персонажем. А если появится желание заставить перемещаться сразу несколько блоков, например, составляющих некую фигуру вроде автомобиля или космического корабля пришельцев?

Сделать это намного сложнее, потому что нужно следить за местоположением сразу нескольких блоков. Всякий раз, когда требуется переместить группу блоков, сначала им нужно присвоить тип `AIR` и воссоздать в новых позициях. Если блоков окажется слишком много, фигура будет перемещаться некрасиво и медленно.

В модуле `minecraftstuff` есть функция `MinecraftShape`, написанная специально для создания и перемещения фигур. Сформированные с ее помощью фигуры сами следят за своими блоками; при перемещении они меняют лишь те блоки, которые действительно изменились.

Чтобы задействовать `MinecraftShape`, надо сообщить, как будет выглядеть фигура, создав список составляющих ее блоков. Каждый блок в фигуре имеет координаты (*x*, *y*, *z*) и тип.

На рис. 7.4 показана простая фигура деревянного коня, состоящая из семи блоков, вместе с их координатами. В данном случае понятие «координаты» не совпадает с понятием координат в мире Minecraft. Взглянув на рис. 7.4, можно убедиться, что блок в центре фигуры имеет координаты 0,0,0, а остальные блоки — относительно центра, то есть блок справа от центра имеет координаты 1,0,0.

Теперь создайте программу, которая будет использовать `MinecraftShape` для создания фигуры деревянного коня, изображенного на рис. 7.4, и перемещать ее:

1. Откройте редактор IDLE и создайте новую программу, выбрав в меню пункт File ▸ New File (Файл ▸ Создать файл). Сохраните файл с именем *WoodenHorse.py* в папке *MyAdventures*.
2. Импортируйте модули `minecraft`, `block`, `minecraftstuff` и `time`:

```
import mcpi.minecraft as minecraft
import mcpi.block as block
import mcpi.minecraftstuff as minecraftstuff
import time
```

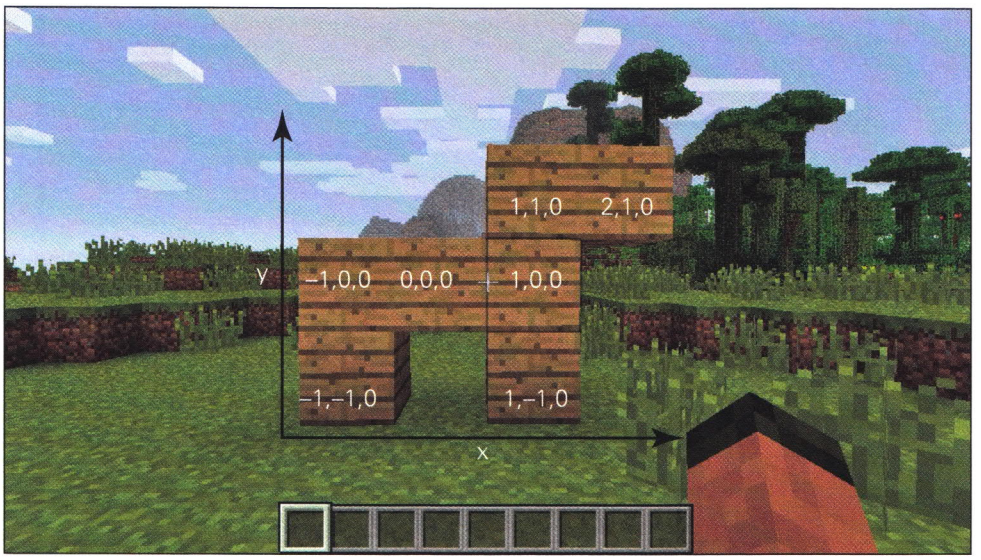


Рис. 7.4. Координаты блоков, образующих фигуру деревянного коня

3. Создайте объект `Minecraft`:

```
mc = minecraft.Minecraft.create()
```

4. Сообщите `MinecraftShape`, где будет создан деревянный конь. Получите позицию персонажа и добавьте единицу к координатам *z* и *y*, так чтобы конь не находился прямо над персонажем:

```
horsePos = mc.player.getTilePos()
horsePos.z = horsePos.z + 1
horsePos.y = horsePos.y+ 1
```

Создайте деревянного коня с помощью `MinecraftShape`, передав объект Minecraft и позицию, где должна быть создана фигура, в качестве параметров:

```
horseShape = minecraftstuff.MinecraftShape(mc, horsePos)
```

5. Настройте блоки коня:

```
horseShape.setBlock(0,0,0,block.WOOD_PLANKS.id)

horseShape.setBlock(-1,0,0,block.WOOD_PLANKS.id)
horseShape.setBlock(1,0,0,block.WOOD_PLANKS.id)
horseShape.setBlock(-1,-1,0,block.WOOD_PLANKS.id)
horseShape.setBlock(1,-1,0,block.WOOD_PLANKS.id)
horseShape.setBlock(1,1,0,block.WOOD_PLANKS.id)
horseShape.setBlock(2,1,0,block.WOOD_PLANKS.id)
```

Позиции блоков такие же, как показано на рис. 7.4.

6. Сохраните и запустите программу. Вуаля! Перед персонажем должен появиться деревянный конь.

7. Добавьте в конец программы `WoodenHorse.py` следующие строки, чтобы она перемещала коня:

```
for count in range(1,10):
  time.sleep(1)
  horseShape.moveBy(1,0,0)
horseShape.clear()
```

Функция `moveBy(x,y,z)` предлагает фигуре «переместиться на» (move by) число блоков, указанное в параметрах `x`, `y`, `z`. В данном примере `horseShape` будет перемещаться на один блок по оси *x*. Функция `clear()` удалит фигуру, присвоив всем ее блокам тип `AIR`.

8. Сохраните, запустите программу и посмотрите, как поскачет деревянный конь!

Кроме функций `moveBy(x,y,z)` и `clear()`, есть функция `move(x,y,z)`, которая переместит фигуру в любую указанную позицию в мире Minecraft. Если фигура удалена функцией `clear()`, ее можно воссоздать, вызвав функцию `draw()`.

Полный код программы, создающей деревянного коня, можно загрузить на сайте книги www.wiley.com/go/adventuresinminecraft2e.

Далее вы убедитесь, насколько удобны фигуры, потому что вам предстоит построить космический корабль пришельцев. А позднее вы снова используете фигуры для создания препятствий.

Вторжение инопланетян

Инопланетяне планируют вторгнуться в мир Minecraft. Космический корабль пикирует сверху прямо на персонажа, который находится в смертельной опасности: инопланетяне настроены враждебно и не успокоятся, пока не добьются своего.

В следующей программе вы будете использовать `MinecraftShape` и приемы программирования, уже использовавшиеся при создании блокфренда, для строительства инопланетного космического корабля (рис. 7.5), который будет парить в воздухе, преследуя персонажа и пытаясь оказаться над ним. Когда это произойдет, он телепортирует персонажа внутрь.

Космический корабль создается с помощью `MinecraftShape`, как деревянный конь в предыдущей программе; каждый его блок будет иметь свои относительные координаты и тип. На рис. 7.6 показаны координаты блоков (если смотреть сбоку и сверху).

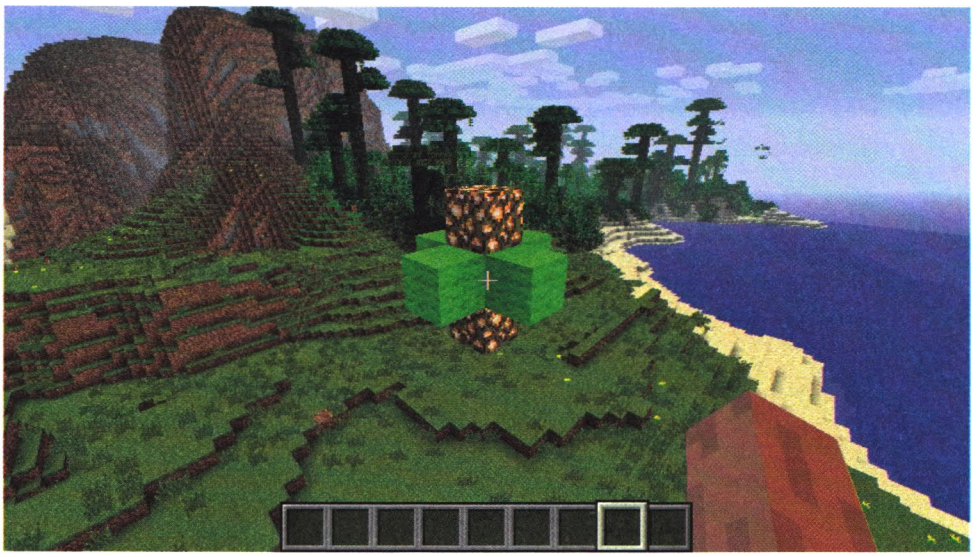


Рис. 7.5. Создайте космический корабль, чтобы придать остроту вашей игре в Minecraft

Рис. 7.6. Координаты блоков, составляющих космический корабль инопланетян

На примере космического корабля инопланетян показано, как с помощью объекта `MinecraftShape` создавать трехмерные фигуры. Они могут быть большими и маленькими, простыми и сложными — какими захотите. Этот объект дает массу возможностей по использованию фигур в программах для Minecraft.

Как и программа блокфренда, программа вторжения инопланетян будет состоять из двух частей. В первой описаны правила, определяющие действия космического корабля, а вторая часть кода перемещает космический корабль в направлении персонажа.

Преследуя персонажа, инопланетяне будут выводить в чат угрозы (например: «Вы не сможете убегать вечно!»). Такие сообщения выбираются из списка случайным образом (рис. 7.7).

Рис. 7.7. Космический корабль инопланетян преследует персонажа

Правила, определяющие дальнейшие действия космического корабля, зависят от одного из трех его состояний:

- **Посадка**: в момент запуска программы космический корабль получит это состояние и начнет снижаться прямо над персонажем.
- **Нападение**: как только корабль сядет на землю, он сразу начнет охоту — будет преследовать персонажа, пока не окажется прямо над ним, после чего телепортирует его внутрь.
- **Задание выполнено**: этот режим устанавливается после того, как персонаж окажется внутри корабля и инопланетяне будут готовы вернуть его на землю. В этой точке программа завершится, а персонаж окажется телепортирован на землю.

Когда инопланетяне захватят персонажа, программа построит мрачную камеру и изменит позицию персонажа так, чтобы он в ней оказался (рис. 7.8). Затем, перед освобождением,

инопланетяне выведут сообщение в чат, вновь изменят позицию персонажа, вернув его на место, и удалят камеру.

Рис. 7.8. Внутри космического корабля инопланетян

Выполните следующие шаги, чтобы создать программу вторжения инопланетян:

1. Откройте редактор IDLE, создайте новую программу, выбрав в меню пункт File ▸ New File (Файл ▸ Создать файл), и сохраните ее с именем *AlienInvasion.py* в папке *MyAdventures*.
2. Импортируйте модули `minecraft`, `block`, `minecraftstuff` и `time`:

```
import mcpi.minecraft as minecraft
import mcpi.block as block
import mcpi.minecraftstuff as minecraftstuff
import time
```

3. Создайте функцию `distanceBetweenPoints()`:

```
def distanceBetweenPoints(point1, point2):
  xd = point2.x - point1.x
  yd = point2.y - point1.y
  zd = point2.z - point1.z
  return math.sqrt((xd*xd) + (yd*yd) + (zd*zd))
```

4. Добавьте константы для программы. `HOVER_HEIGHT` — высота (в блоках) над персонажем, на которой должен оказаться космический корабль; `ALIEN_TAUNTS` — список угроз, отправляемых пришельцами в чат во время преследования персонажа:

```
HOVER_HEIGHT = 15
ALIEN_TAUNTS = ["<aliens>You cant run forever",
                "<aliens>Resistance is useless",
                "<aliens>We only want to be friends"]
```

Вы можете изменить текст угроз: посмотрим, насколько вы изобретательны! Или добавить новые.

5. Создайте объекты `Minecraft` и `MinecraftDrawing`:

```
mc = minecraft.Minecraft.create()
mcdrawing = minecraftstuff.MinecraftDrawing(mc)
```

6. Определите начальное местоположение космического корабля пришельцев на 50 блоков выше персонажа и его режим `"landing"` (посадка):

```
alienPos = mc.player.getTilePos()
alienPos.y = alienPos.y + 50
mode = "landing"
```

7. Создайте космический корабль с помощью `MinecraftShape` (взгляните на рис. 7.6, чтобы вспомнить, как это делается):

```
alienShape = minecraftstuff.MinecraftShape(mc, alienPos)

alienShape.setBlock(-1,0,0,block.WOOL.id, 5)
alienShape.setBlock(0,0,-1,block.WOOL.id, 5)
alienShape.setBlock(1,0,0,block.WOOL.id, 5)
alienShape.setBlock(0,0,1,block.WOOL.id, 5)
alienShape.setBlock(0,-1,0,block.GLOWSTONE_BLOCK.id)
alienShape.setBlock(0,1,0,block.GLOWSTONE_BLOCK.id)
```

8. Создайте цикл `while`, выполняемый, пока не будет установлен режим `"missionaccomplished"` (задание выполнено) космического корабля или, говоря другими словами, пока цикл не прервет выполнение, как только будет установлен режим `"missionaccomplished"`:

```
while mode != "missionaccomplished":
```

9. В каждой итерации цикла определите позицию персонажа:

```
  playerPos = mc.player.getTilePos()
```

10. Следующий фрагмент кода имеет отношение к правилам, используемым программой, чтобы определить, что делать дальше. Если космический корабль находится в режиме `"landing"` (посадка), установите для него цель (куда он должен двигаться), чтобы оказаться над персонажем, и режим `"attack"` (нападение):

```
  if mode == "landing":
    mc.postToChat("<aliens> We dont come in peace - ↩
      please panic")
    alienTarget = playerPos.clone()
    alienTarget.y = alienTarget.y + HOVER_HEIGHT
    mode = "attack"
```

11. В другом случае, если установлен режим `"attack"` (нападение), проверьте, не находится ли космический корабль над персонажем. Если это так, телепортируйте его внутрь корабля и установите режим `"missionaccomplished"` (задание выполнено). Иначе — если персонаж убежал — установите цель в текущую позицию персонажа и выведите в чат угрозу:

```
    elif mode == "attack":
      # проверить, находится ли корабль инопланетян над персонажем
      if alienPos.x == playerPos.x and alienPos.z == ↩
        playerPos.z:
        mc.postToChat("<aliens>We have you now!")

        # создать камеру
        mc.setBlocks(0,50,0,6,56,6,block.BEDROCK.id)
        mc.setBlocks(1,51,1,5,55,5,block.AIR.id)
        mc.setBlock(3,55,3,block.GLOWSTONE_BLOCK.id)

        # телепортировать персонажа
        mc.player.setTilePos(3,51,5)
        time.sleep(10)
        mc.postToChat("<aliens>Not very interesting at all - ↩
         send it back")
        time.sleep(2)

        # вернуть персонажа туда, откуда он был телепортирован
        mc.player.setTilePos(
          playerPos.x, playerPos.y, playerPos.z)

        # удалить камеру
        mc.setBlocks(0,50,0,6,56,6,block.AIR.id)
        mode = "missionaccomplished"

      else:
        # персонаж убежал
        mc.postToChat(
          ALIEN_TAUNTS[random.randint(0,len(ALIEN_TAUNTS)-1)])
        alienTarget = playerPos.clone()
        alienTarget.y = alienTarget.y + HOVER_HEIGHT
```

Когда персонаж телепортируется на борт космического корабля, на высоте 50 блоков над ним создается камера, и позиция персонажа меняется — он оказывается внутри нее. (Как Тардис, машина времени доктора Кто[1]: «Just like Doctor Who's Tardis» («Внутри больше, чем снаружи»).) Затем в чат выводится сообщение, персонаж телепортируется обратно на место, где был захвачен, и камера уничтожается.

Камера, куда телепортируется персонаж, создается в воздухе над ним отчасти потому, что так удобнее, отчасти — поскольку в этом месте наверняка не окажется ничего другого. Однако вы можете создать ее в любом другом месте. Ведь как только персонаж покинет камеру, она исчезнет и мир Minecraft вновь станет таким, каким был.

[1] https://ru.wikipedia.org/wiki/ТАРДИС. — *Примеч. пер.*

12. Если позиция комического корабля не совпадает с целью (устанавливается в правилах выше), направьте его к цели, введя следующий код в конце цикла `while`:

```
if alienPos != alienTarget:

  line = mcdrawing.getLine(
    alienPos.x, alienPos.y, alienPos.z,
    alienTarget.x,alienTarget.y,alienTarget.z)

  for nextBlock in line:
    alienShape.move(
      nextBlock.x, nextBlock.y, nextBlock.z)

    time.sleep(0.25)

  alienPos = alienTarget.clone()
```

13. В этой точке выполнение вернется в начало цикла `while`. Когда режим космического корабля станет `"missionaccomplished"` (задание выполнено) и цикл `while` завершится, последняя строка в программе сотрет корабль инопланетян:

```
alienShape.clear()
```

14. Сохраните и запустите программу и не позвольте инопланетянам оказаться прямо над вами!

Полный код программы вторжения инопланетян можно загрузить на сайте книги www.wiley.com/go/adventuresinminecraft2e.

УГЛУБЛЯЕМСЯ В КОД

Всякий раз, когда выясняется, что персонаж сбежал от инопланетян, в чат выводится случайная угроза из константы `ALIEN_TAUNTS`.

```
mc.postToChat(ALIEN_TAUNTS[random.randint(0,len(ALIEN_TAUNTS)-1)])
```

`ALIEN_TAUNTS` — список строк; функция `random.randint()` используется здесь, чтобы получить индекс случайного элемента списка: число от 0 до количества элементов в списке `ALIEN_TAUNTS` минус 1.

Вычесть 1 необходимо, потому что `len()` возвращает фактическое количество элементов в списке (в данном случае три), а индексация элементов в списке начинается с 0 (список содержит элементы с индексами 0, 1 и 2).

Космический корабль инопланетян в действительности слишком прост. Попробуйте создать более интересный вариант. Измените поведение корабля так, чтобы после приземления он переходил в режим «lurk» (засада), в котором будет находиться неподалеку от персонажа, не нападая, а затем, основываясь на вероятности, без предупреждения переключался в режим нападения.

Дополнительные приключения в моделировании

В этом Приключении вы использовали алгоритмы и правила для моделирования поведения друга и недружественных инопланетян. А теперь подумайте, как можно смоделировать поведение стаи птиц (то есть блоков), океанских волн в мире Minecraft или замкнутой клеточной системы, как в игре «Жизнь», которую придумал английский математик Джон Конвей (John Horton Conway).

Подробнее об игре «Жизнь» можно прочитать на странице https://ru.wikipedia.org/wiki/Жизнь_(игра).

Краткая справочная таблица

Команды	Описание
`import random`	Импортирует модуль `random`
`random.randint(start, end)`	Возвращает случайное число в диапазоне между числами `start` и `end`
`import minecraftstuff`	Импортирует модуль `minecraft`, включенный в начальный комплект инструментов
`mc.getHeight(x,z)`	Возвращает высоту (координату `y`) поверхности земли в точке с координатами `x` и `z` (то есть высоту первого блока, если смотреть сверху вниз, тип которого отличается от `AIR`)
`mcdrawing = minecraftstuff. ↩ MinecraftDrawing(mc)`	Создает объект `MinecraftDrawing`
`copyOfPosition = position.clone()`	Создает копию координат в мире Minecraft
`mcdrawing.getLine(x1, y1, z1, x2, y2, z2)`	Возвращает все блоки, находящиеся на линии, соединяющей две точки

`shape = minecraftstuff. ↩ MinecraftShape(mc, pos)`
`shape.setBlock(1,0,0,block.DIRT.id)`
`shape.setBlock(0,0,1,block.DIRT.id)`
	Создает фигуру `MinecraftShape`

`shape.moveBy(x,y,z)`	Перемещает фигуру `MinecraftShape` на расстояния по координатам, указанным в параметрах `x`, `y` и `z`

Краткая справочная таблица	
Команды	**Описание**
`shape.move(x,y,z)`	Перемещает фигуру `MinecraftShape` в позицию с координатами `x`, `y` и `z`
`shape.clear()`	Удаляет фигуру `MinecraftShape`
`shape.draw()`	Воссоздает фигуру `MinecraftShape`

Новое достижение: похищенный искусственным интеллектом, созданным собственными руками.

В СЛЕДУЮЩЕМ ПРИКЛЮЧЕНИИ

В следующем Приключении вы используете все свои навыки, полученные в предыдущих Приключениях, для создания игры, в которой игрок должен собирать алмазы на время. Но будьте начеку: в этой игре на вашем пути будет возникать множество препятствий!

Приключение 8
Большое приключение и коварные препятствия

Вы достигли последнего большого приключения! Теперь вы используете все навыки, полученные в путешествии, чтобы создать настоящую мини-игру в Minecraft. Вы узнаете, насколько универсален мир Minecraft и как добиться чего-то необычного, использовав простые команды получения и изменения позиций блоков и персонажа.

Также вы приобретете новые навыки и научитесь приемам *многопоточного программирования* (*threading*), чтобы программа могла решать несколько задач одновременно.

Этот проект можно продолжить расширять — завершение работы над ним не означает конец приключения: вы подойдете к точке, из которой начинается путь к созданию еще более красочных, сложных и увлекательных игр.

Игра внутри игры

Игра, которую вам предстоит написать, называется «Crafty Crossing» («Коварные препятствия»). Ее цель — собрать алмазы и попасть на другую сторону игрового поля до того, как истечет отведенное время, постоянно перемещаясь между разными препятствиями, замедляющими ваше продвижение.

За каждый подобранный алмаз начисляются очки, а когда персонаж достигает противоположной стороны, заработанные очки умножаются на число неизрасходованных секунд.

Чтобы пересечь игровое поле в кратчайший срок, персонажу придется запрыгивать на движущиеся платформы, пролезать под стенами, которые двигаются вверх-вниз, и избегать ям-ловушек, разбросанных по игровому полю в случайном порядке (рис. 8.1).

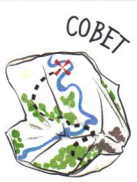
На сайте книги www.wiley.com/go/adventuresinminecraft2e можно найти видеоурок, где показано, как создать игру «Коварные препятствия».

Рис. 8.1. Игра «Коварные препятствия»

Главной целью этого Приключения является создание игры. Вы должны сконструировать, проверить и объединить все компоненты игры, прежде чем она будет готова к использованию. Чтобы было проще следовать за проектом, мы разбили инструкции на три части:

- **часть 1**: создание основного каркаса программы и конструирование игрового поля, где будут происходить все события;
- **часть 2**: программирование препятствий на пути персонажа, чтобы замедлить его продвижение;
- **часть 3**: объединение всего, что было создано прежде, в готовую игру; формирование уровней сложности, учет заработанных очков и неизбежный финал «Game Over» («Конец игры»).

Закончив игру, вы сможете продолжить ее расширять в любом направлении, используя для этого весь свой творческий потенциал.



В каждой части Приключения есть задания для самостоятельного выполнения. Отложите их на потом, когда программа будет закончена, и не вносите в нее никаких изменений, иначе вы рискуете запутаться. Вернитесь к заданиям, завершив приключения.

Часть 1. Создание игрового поля

Игровое поле в игре «Коварные препятствия» — место, где будут происходить все игровые события. Прохождение поля начинается на одной его стороне и заканчивается на другой. При этом на поле присутствует множество препятствий, замедляющих движение вперед.

Без препятствий игровое поле представляет собой прямоугольник из блоков GRASS (трава), огражденный стенами из блоков GLASS (стекло), как показано на рис. 8.2. Ширина, высота и длина поля будут определяться константами. Меняя эти константы, вы можете менять размеры и форму игрового поля; препятствия будут подстраивать свои габариты автоматически. Пол игрового поля должен быть толщиной в три блока, так как на глубине двух блоков начинается водное препятствие.

Рис. 8.2. Вид игрового поля

Помните, как вы использовали функцию `setBlocks()` для строительства дома в Приключении 3? Теперь вы примените ее для строительства игрового поля.

Запустите Minecraft, IDLE и сервер. У вас должно быть достаточно практических навыков, чтобы выполнить эти операции самостоятельно, но если понадобится освежить их в памяти, загляните в Приключение 1.

Создайте новую программу для игры «Коварные препятствия»; начните с конструирования каркаса игры.

1. Откройте редактор IDLE, создайте в папке *MyAdventures* новый файл с именем *CraftyCrossing.py*.
2. Импортируйте необходимые модули. В этой программе используется новый для вас модуль `thread`, описанный во второй части, где рассказывается о создании препятствий:

```
import mcpi.minecraft as minecraft
import mcpi.block as block
import mcpi.minecraftstuff as minecraftstuff
import time
import random
import thread
```

3. Создайте три константы, определяющие размеры игрового поля, — ширину (x), высоту (y) и длину (z):

```
ARENAX = 10
ARENAZ = 20
ARENAY = 3
```

4. Добавьте заготовки функций. Вы будете создавать их на протяжении всего приключения, пока игра не закончится:

```
def createArena(pos):
  pass
def theWall(arenaPos, wallZPos):
  pass
def theRiver(arenaPos, riverZPos):
  pass
def theHoles(arenaPos, holesZPos):
  pass
def createDiamonds(arenaPos, number):
  pass
```

Инструкция `pass` ничего не делает, но она может стать отличным «заместителем» программного кода, который будет создан потом.

5. Подключитесь к игре Minecraft:

```
mc = minecraft.Minecraft.create()
```

6. Создайте логическую (булеву) переменную со значением `True` по окончании игры и `False` — в начале:

```
gameOver = False
```

По мере движения вперед вы будете добавлять новый код в основную программу либо в только что созданные функции.

Прямо сейчас можно запустить программу. Но пока она ничего не делает и на экране ничего не происходит! Тем не менее есть смысл запустить программу в ее текущем виде, чтобы убедиться в отсутствии ошибок, которые иначе могут появиться в интерактивной оболочке Python.

Следующий шаг — изменить функцию `createArena`, которая строит игровое поле в мире Minecraft.

1. Найдите функцию `createArena` в только что созданном каркасе программы:

```
def createArena(pos):
  pass
```

и удалите инструкцию `pass`.

2. Ниже строки `def createArena(pos):` с одним отступом добавьте создание соединения с игрой Minecraft:

```
  mc = minecraft.Minecraft.create()
```

3. Теперь можно создать игровое поле с помощью `setBlocks`, взяв координаты `(pos)`, переданные в функцию, и сложив их с константами `ARENAX`, `ARENAY` и `ARENAZ` (на рис. 9.3 показано, как константы складываются с координатами для создания поля):

```
mc.setBlocks(pos.x - 1 , pos.y, pos.z - 1,
             pos.x + ARENAX + 1, pos.y - 3,
             pos.z + ARENAZ + 1,
             block.GRASS.id)
```

4. Далее постройте стеклянную стену, создав куб из блоков `GLASS`, и очистите внутреннее пространство, создав куб из блоков `AIR` внутри:

```
mc.setBlocks(pos.x - 1, pos.y + 1, pos.z - 1,
             pos.x + ARENAX + 1, pos.y + ARENAY,
             pos.z + ARENAZ + 1,
             block.GLASS.id)
mc.setBlocks(pos.x, pos.y + 1, pos.z,
             pos.x + ARENAX, pos.y + ARENAY,
             pos.z + ARENAZ,
             block.AIR.id)
```

5. Теперь функция `createArena()` готова к использованию, но ее нужно вызвать из главной программы. Добавьте в конец программы следующие строки, которые определяют позицию персонажа и передают ее через переменную в вызов функции `createArena()`:

```
arenaPos = mc.player.getTilePos()
createArena(arenaPos)
```

6. Сохраните и запустите программу. Вы должны увидеть, как перед персонажем появится игровое поле (рис. 8.3).

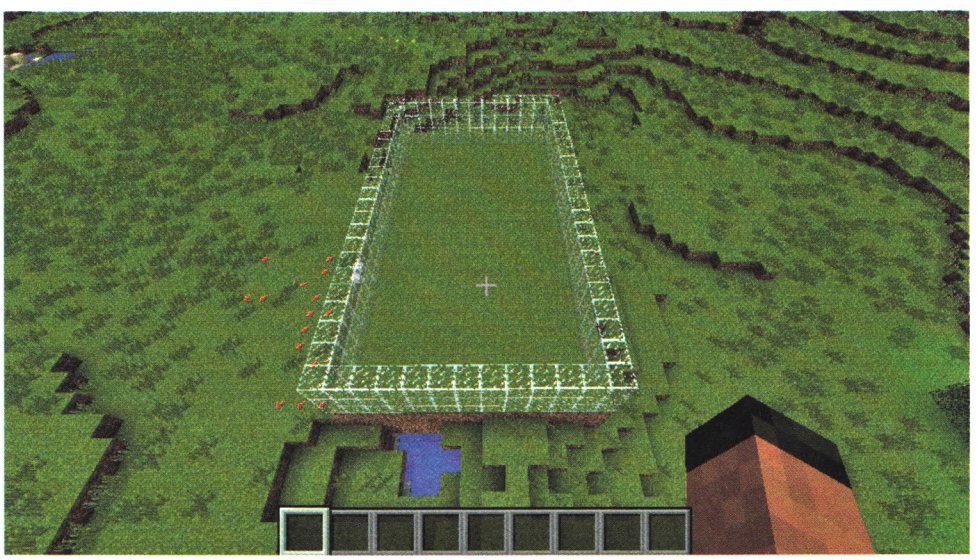


Рис. 8.3. Игровое поле создано

Игровое поле создано, но пока выглядит скучным. Можно ли сделать его интереснее? Пожалуй, можно добавить стрелки, обозначающие направление, куда должен идти персонаж, или какие-нибудь декорации по бокам. Или, может, установить факелы на крыше?

Часть 2. Создание препятствий

Дополнительные сложности и препятствия делают игру более интересной — простая игра быстро надоедает. В следующей части Приключения вы создадите несколько препятствий, чтобы замедлить продвижение персонажа и усложнить достижение противоположного края игрового поля.

Стена

Первое препятствие, которое вы создадите, — кирпичная стена. Она не просто пресекает игровое поле по всей ширине, но и двигается вверх-вниз. Когда стена находится внизу, персонажу приходится ждать, пока стена поднимется на достаточную высоту, чтобы пройти под ней (рис. 8.4).

Рис. 8.4. Стена может замедлить продвижение персонажа

Стена — простой, но эффективный способ замедлить продвижение персонажа. Однако если играющий правильно все рассчитает, он сможет провести персонажа под стеной и приступить к преодолению следующего препятствия.

Вы создадите стену с помощью объекта `MinecraftShape` в модуле `minecraftstuff`, с которым познакомились, когда программировали космический корабль инопланетян в Приключении 7.

Измените функцию `theWall()`, создающую стену, выполнив следующие шаги:

1. Найдите функцию в каркасе программы по следующим строкам:

```
def theWall(arenaPos, wallZPos):
  pass
```

   и удалите инструкцию `pass`.

2. Ниже строки `the def theWall(arenaPos, wallZPos):` с одним отступом добавьте создание соединения с игрой Minecraft:

```
  mc = minecraft.Minecraft.create()
```

3. Функция `theWall()` принимает два параметра: `arenaPos` (позиция игрового поля) и `wallZPos` (смещение стены от начала игрового поля вдоль оси *z*). Используя эти два параметра, определите позицию стены (рис. 8.5):

```
  wallPos = minecraft.Vec3(arenaPos.x,
                           arenaPos.y + 1,
                           arenaPos.z + wallZPos)
```

4. Создайте фигуру для стены, передав в функции `mc` и `wallPos` переменные, которые были созданы на шаге 2 и 3:

```
wallShape = minecraftstuff.MinecraftShape(mc, wallPos)
```

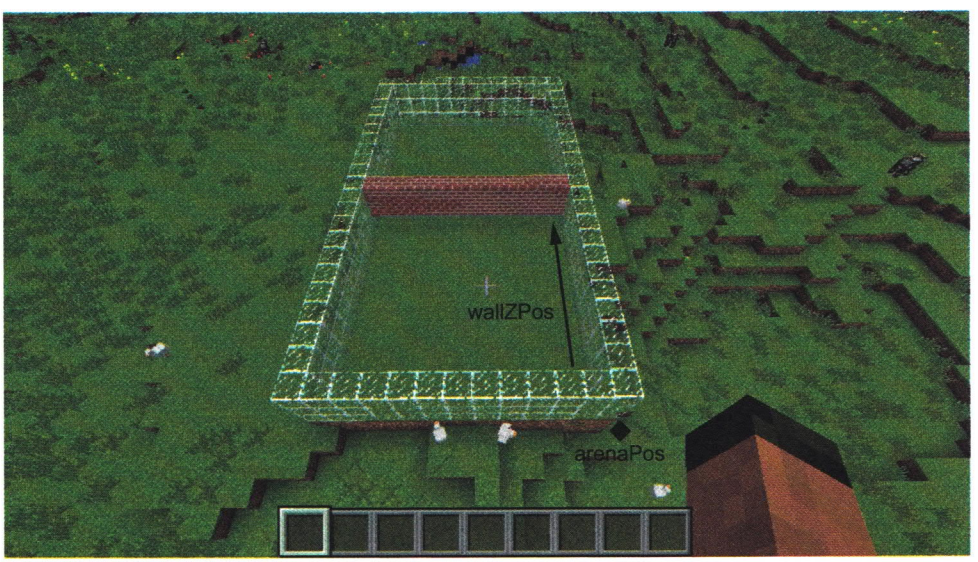


Рис. 8.5. Определение позиции создания стены

5. Теперь создайте фигуру для стены с помощью `setBlocks` для генерирования блоков по арене (x) и выше (y):

```
wallShape.setBlocks(
      0, 1, 0,
      ARENAX, ARENAY - 1, 0,
      block.BRICK_BLOCK.id)
```

6. Строительство стены завершено, но... она неподвижна! Заставить стену двигаться вверх-вниз можно парой вызовов функции `MinecraftShape moveBy()` с небольшой задержкой между ними:

```
while not gameOver:
  wallShape.moveBy(0,1,0)
  time.sleep(1)
  wallShape.moveBy(0,-1,0)
  time.sleep(1)
```

   Код в цикле `while` будет выполняться, пока переменная `gameOver` не получит значение `True` (пока выполняется условие `not gameOver`). Эта переменная получит значение `True` в конце игры, когда персонаж дойдет до конца или проиграет.

7. Функция `theWall()` готова. Все, что осталось сделать, — вызвать функцию из главной программы. Поэтому добавьте следующие строки в конец программы:

```
WALLZ = 10
theWall(arenaPos, WALLZ)
```

   Константа `WALLZ` хранит смещение относительно начала игрового поля вдоль оси Z, где должна быть построена стена.

8. Сохраните и запустите программу. Вы должны увидеть стену, двигающуюся вверх-вниз в середине игрового поля.

Вы еще не написали программный код, присваивающий переменной `gameOver` значение `True`. Поэтому когда запустите программу, она будет выполняться до бесконечности. Вам придется остановить ее вручную, выбрав пункт Shell ▸ Restart Shell (Оболочка ▸ Перезапустить оболочку) или удерживая Ctrl+C в меню интерактивной оболочки Python.

УГЛУБЛЯЕМСЯ В КОД

Позиция фигуры `theWall` создается с помощью `minecraft.Vec3 (x, y, z)`. `Minecraft.Vec3 ()`— это способ API Minecraft по объединению множества координат (x, y, z). `Vec3` является самым быстрым для трехмерного вектора.

ПОДДЕРЖКА НЕСКОЛЬКИХ ДЕЙСТВУЮЩИХ ПРЕПЯТСТВИЙ

Теперь у нас появилась проблема: программа зациклилась. Она будет крутиться на одном месте, перемещая стену вверх-вниз и ничего другого не делая. Это объясняется тем, что программа написана в **последовательном** стиле, то есть все ее команды выполняются по очереди и каждая следующая команда ждет, пока выполнится предыдущая. Программа зациклится, потому что не сможет выполнить следующую команду после цикла `while`.

Слово **последовательно** означает следование в определенном порядке, когда одно идет за другим.

Но как написать несколько действующих препятствий и обеспечить работу оставшейся части программы, если она зацикливается и не хочет делать ничего другого, только двигать стену вверх-вниз?

Это проблема имеет решение. Встречайте многопоточное выполнение! До настоящего момента все написанные вами программы были однопоточными — команды в них выполнялись последовательно, одна за другой.

Если представить программу как бечевку с узелками (команды), можно представить ее выполнение как движение вдоль бечевки от начала до конца и выполнение очередной команды, когда встречается узелок. Если добавить в программу цикл, он будет заставлять ее возвращаться к предыдущим узелкам, а если добавить инструкцию `if`, она может заставить программу пропустить какие-то узелки. Однако программа все равно будет выполнять не более одной команды в каждый момент времени.

Используя многопоточность, вы предлагаете своей программе создать новый поток выполнения (добавить еще одну бечевку) со своими командами (узелками). Эти команды будут выполняться параллельно основной программе, которая тоже продолжит выполняться. Теперь ваша программа будет решать две задачи вместо одной (рис. 9.6).

Чтобы все препятствия в игре могли работать одновременно, следует создать новый поток выполнения для каждого действующего препятствия, чтобы основная программа и препятствия действовали одновременно.

Многопоточный режим выполнения чрезвычайно удобен в компьютерных программах, но это сложный прием, и его сложность быстро возрастает. Желающие ближе познакомиться с режимом многопоточного выполнения в Python могут посетить страницу www.tutorialspoint.com/python/python_multithreading.htm[1].

[1] Отличную статью на русском языке можно найти по адресу: http://www.plam.ru/compinet/jazyk_programmirovanija_python/p12.php. — *Примеч. пер.*

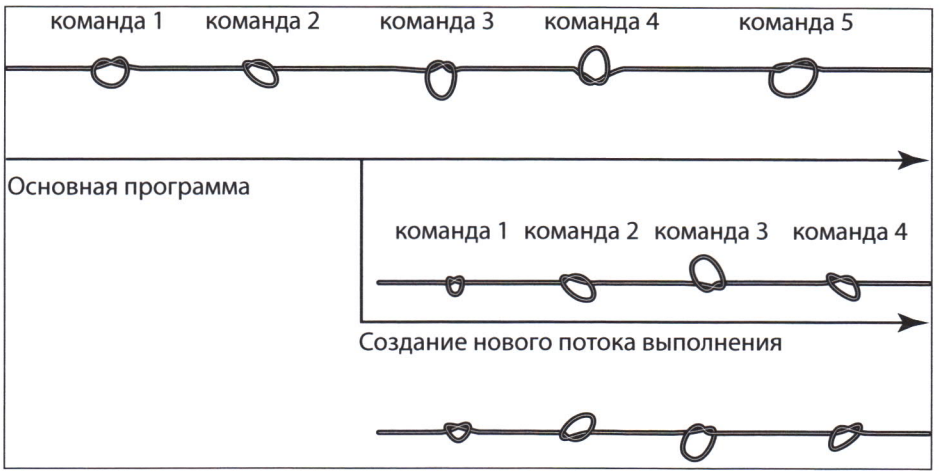


Рис. 8.6. Создавая несколько потоков выполнения, программа получает возможность решать несколько задач одновременно

А теперь вернемся к нашей стене. Чтобы организовать действие стены в отдельном потоке, нужно изменить строку программы, где вызывается функция `theWall()`, и использовать функцию `thread.start_new_thread()`:

1. Удалите в программе последнюю строку, вызывающую функцию `theWall()`, которая выглядит так:

```
theWall(arenaPos, WALLZ)
```

2. Вставьте следующую строку в конце программы, чтобы создать новый поток `wall_t`, который вызовет функцию `theWall()`:

```
wall_t = threading.Thread(
    target = theWall,
    args = (arenaPos, WALLZ))
```

3. Теперь необходимо запустить поток `wall`:

```
wall_t.start()
```

4. Сохраните и запустите программу.

Вы должны увидеть тот же результат, что и прежде, со стеной поперек игрового поля, двигающейся вверх-вниз. Отличие данной версии программы в том, что теперь движение стены выполняется в отдельном потоке. Теперь вы можете продолжить программирование оставшейся части игры.

Если вы захотите остановить многопоточную программу, запущенную в IDLE, выберите пункт Shell ▸ Restart Shell (Оболочка ▸ Перезапустить оболочку) в меню интерактивной оболочки Python. Если просто нажать комбинацию клавиш Ctrl+C, остановится лишь основная программа, а другие потоки продолжат выполнение. Поэтому остановка многопоточной программы должна осуществляться именно выбором пункта меню Shell ▸ Restart Shell (Оболочка ▸ Перезапустить оболочку).

УГЛУБЛЯЕМСЯ В КОД

Чтобы запустить функцию `theWall()` в ее потоке, используйте этот код:

```
wall_t = threading.Thread(
    target = theWall,
    args = (arenaPos, WALLZ))
wall_t.start()
```

Целевому параметру присваивается имя функции, которая является функцией `Wall`. Второй параметр `args` представляет собой переменные, которые функция ожидает передать: `arena`, `WALLZ`.

Переменные (параметры), которые функция ожидает получить, передаются в скобках (`arena`, `WALLZ`), потому что `args` ожидает, что они будут переданы как **кортеж**.

Любая функция Python может быть вызвана таким образом. Если вы написали следующую функцию для печати сообщения на экран

```
def printMessage(message)
  print message
```

вы можете запустить ее в собственном потоке с помощью:

```
print_t = threading.Thread(
    target = printMessage,
    args = ("Hello Minecraft World",))
print_t.start()
```

Запятая после «`Hello Minecraft World`» сообщает Python, что это кортеж, но только с одним элементом внутри него.

Если вы хотите, чтобы ваша основная программа дождалась завершения потока, то можете использовать функцию `join ()`, которая при вызове будет длиться до тех пор, пока не будет запущен весь код в потоке:

```
print_t.join()
```

Кортежи похожи на списки, которые вы использовали в предыдущих Приключениях. Главная особенность кортежей в том, что они не могут изменяться после создания, в отличие от списков, которые можно менять, добавляя в них новые элементы и удаляя существующие. В программировании структуры, которые не могут изменяться, так и называют неизменяемыми, а структуры, способные изменяться, — изменяемыми. Желающие ближе познакомиться с кортежами в языке Python могут посетить страницу www.tutorialspoint.com/python/python_tuples.htm[1].

[1] Аналогичное описание на русском языке можно найти по адресу: http://ru.diveintopython.net/odbchelper_tuple.html. — *Примеч. пер.*

Создание рва

Следующая ваша задача — создать ров, наполненный водой и пересекающий игровое поле по всей ширине. Ров настолько широк, что персонаж не в состоянии его перепрыгнуть! К счастью, через ров можно перейти по специальной платформе, но она двигается вдоль рва взад-вперед, поэтому игрок должен точно рассчитать момент, когда нужно запрыгнуть на платформу и спрыгнуть с нее на другой стороне (рис. 8.7).

Если персонаж упадет в ров, он должен быть возвращен в начало игрового поля. Программный код, возвращающий персонажа в начало, вы напишете в части 3, а пока займемся строительством рва и движущейся платформы.

Рис. 8.7. Персонаж должен пересечь ров

Сначала нужно создать сам ров, удалив часть блоков на поверхности игрового поля и добавив в получившийся ров слой из блоков воды. Создать платформу можно с помощью `MinecraftShape`, а чтобы заставить ее двигаться вдоль рва, можно воспользоваться приемом, который применялся, чтобы заставить двигаться стену вверх-вниз.

Измените функцию `theRiver()`, чтобы создать ров, выполнив следующие шаги:

1. Найдите функцию `theRiver()` по следующим строкам в программе:

```
def theRiver(arenaPos, riverZPos):
  pass
```

   и удалите инструкцию `pass`.

2. Ниже строки `def theRiver(arenaPos, riverZPos):` с одним отступом добавьте создание соединения с игрой Minecraft:

```
  mc = minecraft.Minecraft.create()
```

3. Создайте две константы, определяющие ширину рва (RIVERWIDTH) и ширину платформы (BRIDGEWIDTH):

```
RIVERWIDTH = 4
BRIDGEWIDTH = 2
```

4. Теперь создайте ров, пересекающий игровое поле, использовав параметры, переданные в функцию, arenaPos и riverZPos (смещение рва от начала игрового поля вдоль оси *z*):

```
mc.setBlocks(arenaPos.x,
             arenaPos.y - 2,
             arenaPos.z + riverZPos,
             arenaPos.x + ARENAX,
             arenaPos.y,
             arenaPos.z + riverZPos + RIVERWIDTH - 1,
             block.AIR.id)
mc.setBlocks(arenaPos.x,
             arenaPos.y - 2,
             arenaPos.z + riverZPos,
             arenaPos.x + ARENAX,
             arenaPos.y - 2,
             arenaPos.z + riverZPos + RIVERWIDTH - 1,
             block.WATER.id)
```

Первый вызов функции setBlocks() создает сам ров, заполняя его объем блоками типа AIR (воздух), а второй вызов setBlocks() заполняет ров слоем блоков типа WATER (вода).

5. Определите координаты для строительства платформы. Она должна находиться в середине рва, чтобы персонаж не мог просто шагнуть на платформу, а должен был бы прыгнуть на нее:

```
bridgePos = minecraft.Vec3(arenaPos.x,
                           arenaPos.y,
                           arenaPos.z + riverZPos + 1)
```

6. Создайте фигуру моста, передав функциям mc и bridgePos переменные, которые были заданы на шаге 2 и 5:

```
bridgeShape = minecraftstuff.MinecraftShape(mc, bridgePos)
```

7. Создайте фигуру, которая будет мостом. Это делается так же, как и стена, создаваемая с помощью setBlocks, с мостом (*x*) и шириной реки (*z*):

```
bridgeShape.setBlocks(
    0, 0, 0,
    BRIDGEWIDTH - 1, 0, RIVERWIDTH - 3,
    block.WOOD_PLANKS.id)
```

При создании платформы из значения RIVERWIDTH вычитается число 3, чтобы создать промежуток между платформой и берегами и заставить игрока запрыгивать на платформу и спрыгивать с нее (рис. 8.8).

8. Чтобы заставить платформу двигаться вдоль рва (поперек игрового поля), нужно вычислить расстояние в блоках, которое она должна преодолевать, двигаясь из одного

конца рва в другой. Это расстояние равно ширине игрового поля минус ширина платформы:

```
steps = ARENAX - BRIDGEWIDTH + 1
```

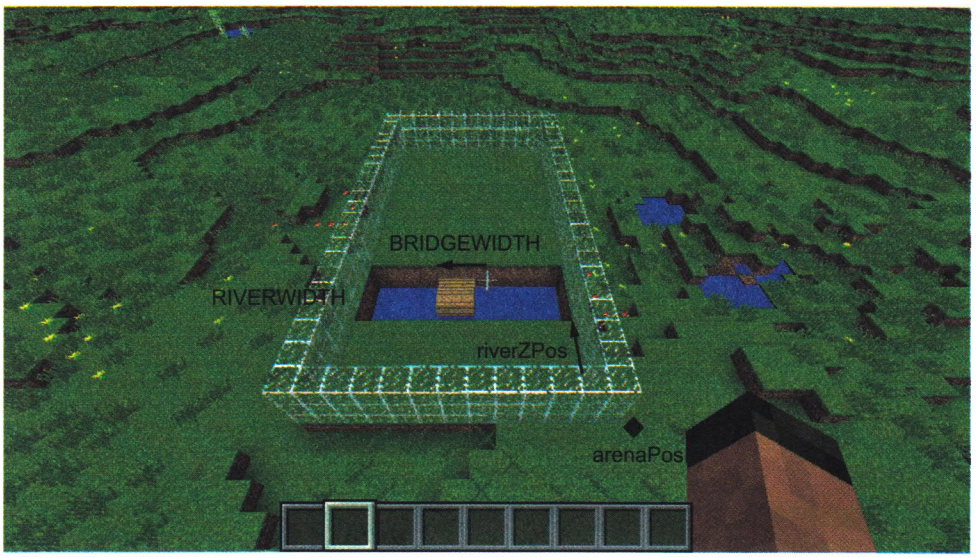


Рис. 8.8. Так как платформа не касается берегов рва, персонаж должен запрыгивать и спрыгивать с нее

9. Перемещение платформы реализовано с помощью двух циклов `for` (один для перемещения влево, другой для перемещения вправо), функции `MinecraftShape.moveBy()`, смещающей платформу на один блок в каждом шаге, и небольшой задержки между шагами:

```
while not gameOver:

  for left in range(0, steps):
    bridgeShape.moveBy(1,0,0)
    time.sleep(1)

  for right in range(0, steps):
    bridgeShape.moveBy(-1,0,0)
    time.sleep(1)
```

10. На этом создание функции `theRiver()` закончено. Теперь ее нужно вызвать в основной программе, в новом потоке выполнения. Для этого добавьте следующий программный код в конец программы:

```
RIVERZ = 4
river_t = threading.Thread(
    target = theRiver,
    args = (arenaPos, RIVERZ))
river_t.start()
```

Константа `RIVERZ` хранит позицию *z* на игровом поле, где будет создано речное препятствие.

11. Самое время запустить программу! Вы должны увидеть игровое поле со стеной в середине и рвом перед ней, вдоль которого взад-вперед плавает платформа.

Попробуйте зайти на игровое поле и преодолеть ров. Если у вас хорошо развита координация движений, это задание не покажется сложным. Но не спешите с выводами! И учтите, что при этом вы будете торопиться собрать алмазы. Вероятность впопыхах промахнуться мимо платформы очень велика.

Создание ям-ловушек

Последнее препятствие, которое нужно создать, — ямы-ловушки. Это обычные ямы; они появляются и исчезают каждые несколько секунд в случайных местах игрового поля (рис. 8.9).

Рис. 8.9. На игровом поле будут появляться ямы-ловушки, и игрок должен быть внимательным, чтобы не попасть в них!

Для выбора случайных позиций вы будете использовать функцию `randint()`. На короткий период времени, перед появлением ям, на поверхности игрового поля будут появляться блоки `BLACK WOOL` (черная шерсть), что послужит игроку предупреждением и даст шанс из-

бежать попадания в яму. Ямы создаются путем преобразования блоков поверхности игрового поля в блоки типа AIR. Спустя несколько секунд после появления ямы будут закрываться, заполняясь блоками типа GRASS, а в других местах будут создаваться новые.

Как и в случае со рвом, попав в яму, персонаж будет возвращаться в начало игрового поля. Но программный код, возвращающий персонажа в начало, вы напишете в части 3 этого Приключения.

Теперь измените функцию theHoles, чтобы она создавала ямы-ловушки:

1. Найдите функцию theHoles()по следующим строкам в программе:

```
def theHoles(arenaPos, holesZPos):
  pass
```

   Удалите инструкцию pass.

2. Ниже строки def theHoles(arenaPos, holesZPos): с одним отступом добавьте создание соединения с игрой Minecraft:

```
  mc = minecraft.Minecraft.create()
```

3. Создайте две константы, определяющие количество ям-ловушек (HOLES) и ширину полосы с ямами (рис. 8.10):

```
HOLES = 15
HOLESWIDTH = 3
```

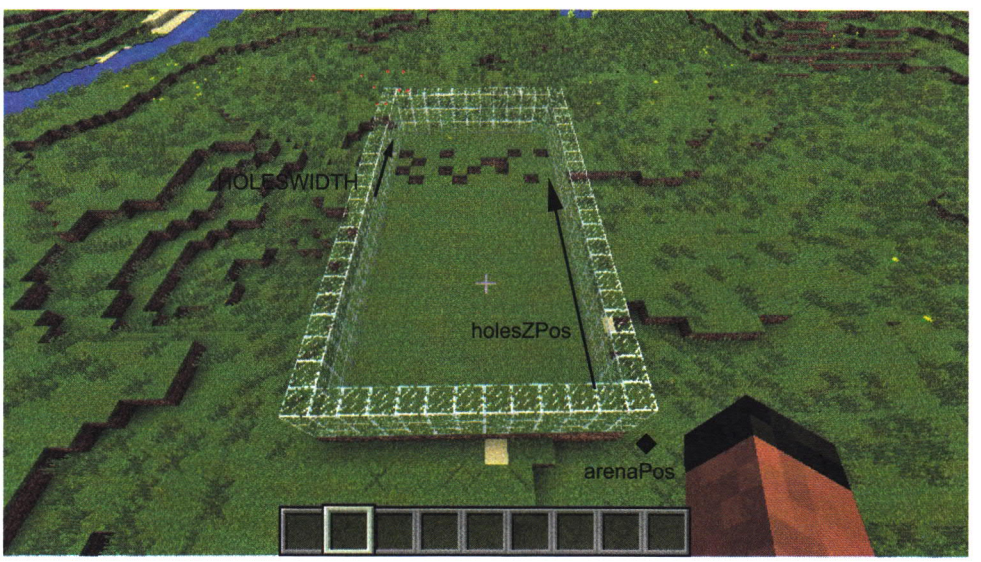


Рис. 8.10. Определите начало полосы ям-ловушек и ее ширину

4. Создайте цикл while, который будет выполняться до завершения программы:

```
while not gameOver:
```

Остальные строки в функции `theHoles` принадлежат циклу `while` и, соответственно, должны иметь два отступа.

5. Определите случайные позиции для ям, вызвав функцию `random.randint()`, чтобы создать координаты *x* и *z* (позиция *y* на позиции игрового поля), и добавьте их в список Python:

```
holes = []
for count in range(0,HOLES):
  x = random.randint(
    arenaPos.x,
    arenaPos.x + ARENAX)

  z = random.randint(
    arenaPos.z + holesZPos,
    arenaPos.z + holesZPos + HOLESWIDTH)

  holes.append(minecraft.Vec3(x, arenaPos.y, z))
```

6. Выполните итерации по всем позициям в списке `holes`, присвойте блокам в этих позициях тип `WOOL` и черный цвет, следующие три цикла `for` следует прописать с отступом под функцией `while not gameOver`:

```
for hole in holes:
  mc.setBlock(hole.x, hole.y, hole.z,
    block.WOOL.id, 15)
time.sleep(0.25)
```

Превращая блоки, которые вскоре станут ямами, в блоки из черной шерсти, вы подсказываете игроку, где появятся ямы, чтобы дать ему шанс обойти ловушки.

7. Откройте ловушки, присвоив блокам в позициях, где должны находиться ямы, тип `AIR`:

```
for hole in holes:
  mc.setBlocks(hole.x, hole.y, hole.z,
               hole.x, hole.y - 2, hole.z,
               block.AIR.id)
time.sleep(2)
```

После создания ям-ловушек выдержите паузу в 2 секунды.

8. Закройте ловушки, используя такой же цикл, как и для их открытия, но на этот раз присвойте блокам тип `GRASS`:

```
for hole in holes:
  mc.setBlocks(hole.x, hole.y, hole.z,
               hole.x, hole.y - 2, hole.z,
               block.GRASS.id)
time.sleep(2)
```

После этого программа вернется в начало цикла `while` и создаст новые ямы-ловушки.

9. На этом создание функции `theHoles` закончено. Добавьте следующий программный код в конец программы:

```
HOLESZ = 15
holes_t = threading.Thread(
    target = theHoles,
    args = (arenaPos, HOLESZ))
holes_t.start()
```

Константа `HOLESZ` хранит координату *z* на игровом поле, где начинается полоса с ямами.

10. Сохраните и запустите программу. Как и ранее, вы должны увидеть игровое поле со стеной и рвом, но на сей раз еще и с полосой ближе к концу игрового поля, где время от времени в случайных местах появляются и исчезают ямы-ловушки.

Попробуйте зайти на игровое поле и пройти его вперед-назад — так, чтобы не попасть в ловушки и не застрять на месте.

Можете подстроить значения констант как посчитаете нужным. Попробуйте сделать игровое поле длиннее или шире, передвинуть препятствия в другие места или усложнить их прохождение, уменьшив задержки.

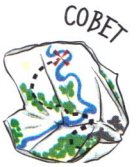
УГЛУБЛЯЕМСЯ В КОД

Функции, реализующие препятствия, действуют подобно маленьким программам, а так как они запускаются в отдельных потоках, то работают независимо от основной программы. Благодаря этому, если вам захочется добавить дополнительные препятствия, вы легко это сделаете. Хотите две стены? Вы их получите, добавив еще один вызов функции `theWall`, но с другой координатой *z*, и на игровом поле возникнет вторая стена, двигающаяся вверх-вниз, как и первая:

```
WALL2Z = 13
wall_t2 = threading.Thread(
    target = theWall,
    args = (arenaPos, WALL2Z))
wall_t2.start()
```

Не останавливайтесь на достигнутом. Используя те же приемы, с помощью которых были созданы стена, ров и ямы-ловушки, создавайте новые виды препятствий. Можно, например, создать клетку, которая будет появляться в случайных местах, пытаясь поймать персонажа, или группу платформ, через которые персонаж должен перепрыгивать, чтобы добраться до противоположного края игрового поля.

Если препятствия покажутся вам слишком простыми, попробуйте изменить значения соответствующих констант, чтобы увеличить их сложность. Например, можно увеличить или уменьшить задержки, чтобы уменьшить или увеличить скорость движения препятствий. Чтобы ускорить движение платформы, замените инструкцию `time.sleep(1)` в циклах `for`, передвигающих платформу влево и вправо, на инструкцию `time.sleep(0.5)`.



Часть 3. Игра

Следующая часть этого большого Приключения — добавление игровой логики в программу. Ваша цель — превратить игровое поле из обычной полосы препятствий в игру, в которую игрок будет хотеть играть снова и снова, переходя с уровня на уровень.

Для этого игра должна быть сложной, захватывающей, приносить вознаграждение и иметь цель.

Цель игры — собрать все алмазы, разбросанные в случайном порядке по игровому полю, и сделать это в заданных временных рамках. Наградой игроку, добравшемуся до конца поля, станут очки за каждый подобранный алмаз. Чем быстрее он пройдет игровое поле, тем больше очков получит. Главная цель игры — пройти все уровни и получить максимальное количество очков.

Игра имеет три уровня, каждый следующий сложнее предыдущего, содержит больше алмазов и на его прохождение дается меньше времени.

Начало игры

В этом разделе вы настроите игру и создадите константы, определяющие количество алмазов и интервал времени для каждого уровня.

Программа состоит из двух основных циклов:

- **Игровой цикл** выполняется до завершения игры (`while not gameOver`). В нем будут подготавливаться и запускаться все уровни, в конце каждого начисляются очки.
- **Цикл прохождения уровня** выполняется до завершения прохождения уровня — до конца уровня или до конца игры (`while not gameOver and not levelComplete`). В этом цикле персонаж будет возвращаться в начало уровня, если попадет в яму-ловушку или упадет в ров, при этом потеряв все собранные алмазы. Здесь же программа следит за временем прохождения поля.

Инструкции, которые вы напишете в этой части Приключения, относятся к главному игровому циклу либо к циклу прохождения уровня. Внимательно делайте отступы, иначе игра будет работать неправильно.

Прежде всего создайте каркас игры и два основных цикла, выполнив следующие шаги и добавив необходимые инструкции в конец программы:

1. Создайте три константы, определяющие количество уровней в игре, алмазов на каждом уровне и интервал времени (в секундах), который дается игроку для прохождения каждого уровня:

```
LEVELS = 3
DIAMONDS = [3,5,9]
TIMEOUTS = [30,25,20]
```

   Константы `DIAMONDS` и `TIMEOUTS` — это списки. Обе имеют по три элемента со значениями для всех трех уровней; так, на первом уровне игрок должен собрать три алмаза и за 30 секунд добраться до противоположной стороны игрового поля.

2. Создайте две переменные для хранения очков, заработанных игроком, и номера текущего уровня:

```
level = 0
points = 0
```

3. Создайте игровой цикл. Добавьте комментарий перед циклом как напоминание. Все следующие инструкции будут относиться к игровому циклу и должны иметь один отступ:

```
# игровой цикл
while not gameOver:
```

   Переменная `gameOver` получает значение `False` в начале программы; когда игрок дойдет до конца последнего уровня или исчерпает отведенное ему время, этой переменной будет присвоено значение `True`. Эта же переменная используется в функциях препятствий: когда она получит значение `True`, все препятствия остановятся.

4. Не забыв про отступы относительно начала игрового цикла, измените позицию персонажа так, чтобы он оказался в начале игрового поля:

```
    mc.player.setPos(arenaPos.x + 1,
                     arenaPos.y + 1,
                     arenaPos.z + 1)
```

5. Запустите часы для текущего уровня, сохранив текущее время в переменной:

```
    start = time.time()
```

6. Установите флаг завершения уровня в значение `False` и создайте цикл прохождения уровня. Как и для игрового цикла, добавьте комментарий, чтобы он напоминал вам, где начинается цикл. Все следующие инструкции будут относиться к циклу прохождения уровня и должны иметь два отступа:

```
    levelComplete = False
    # цикл прохождения уровня
    while not gameOver and not levelComplete:
```

7. Не забывая про отступы относительно начала цикла прохождения уровня, добавьте небольшую задержку. Она необходима, потому что иначе программа будет постоянно выполнять цикл, используя всю вычислительную мощность компьютера:

```
        time.sleep(0.1)
```

8. Сохраните и запустите программу. Единственное изменение, которое вы увидите, — персонаж окажется в начале игрового поля; этот пробный пуск позволит вам убедиться, что все работает правильно и без ошибок.

Вначале, перед прохождением уровня, персонаж оказывается в правом углу игрового поля. Может, было бы лучше, если бы он оказался в середине? Измените программный код так, чтобы в начале каждого уровня персонаж оказывался посередине ближнего края игрового поля.

Сбор алмазов

Главной целью игры является сбор алмазов. Поэтому далее вам предстоит так запрограммировать драгоценные камни, чтобы они появлялись в случайных местах игрового поля (рис. 8.11). Игрок будет собирать алмазы, касаясь их мечом (точнее говоря, щелкая правой кнопкой мыши и удерживая меч на блоке).

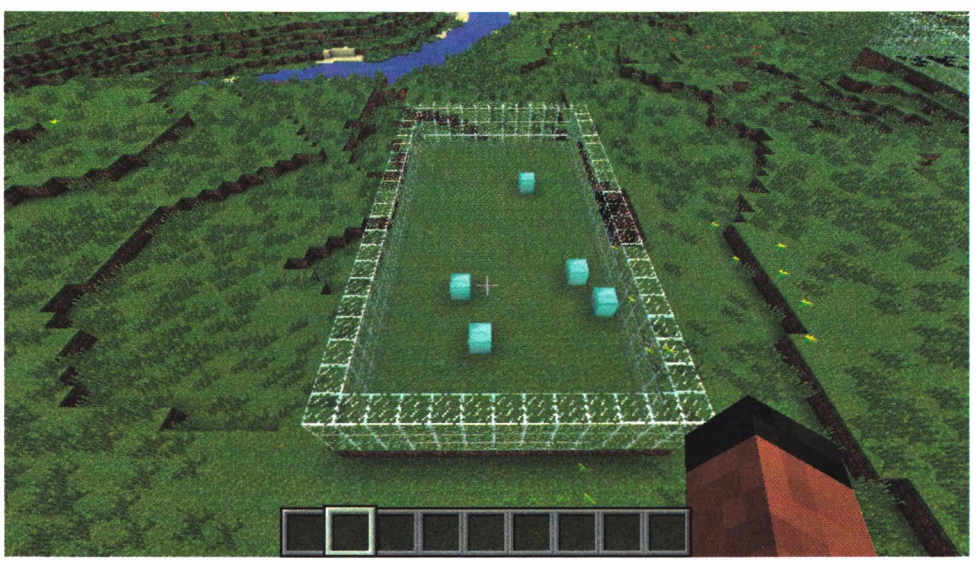


Рис. 8.11. Алмазы появляются в случайных местах игрового поля

Алмазы будут исчезать, как только персонаж их подберет. Собрав все алмазы, игрок может закончить уровень, добравшись до противоположного края игрового поля.

Измените функцию `createDiamonds()` и добавьте ее вызов в игровой цикл, выполнив следующие шаги:

1. Найдите функцию `createDiamonds()` по следующим строкам:

```
def createDiamonds(arenaPos, number):
  pass
```

   Удалите инструкцию `pass`.

2. Ниже строки `def createDiamonds(arenaPos, number):` с одним отступом добавьте создание соединения с игрой Minecraft:

```
  mc = minecraft.Minecraft.create()
```

3. Создайте алмазы в количестве, предусмотренном для данного уровня, установив для блоков со случайными координатами *x* и *z* на игровом поле тип `DIAMOND_BLOCK`:

```
  for diamond in range(0, number):
    x = random.randint(arenaPos.x, arenaPos.x + ARENAX)
    z = random.randint(arenaPos.z, arenaPos.z + ARENAZ)
    mc.setBlock(x, arenaPos.y + 1, z,
                block.DIAMOND_BLOCK.id)
```

4. Теперь функцию `createDiamonds()` нужно вызвать в начале игрового цикла; всякий раз в начале каждого нового уровня будет создаваться новый набор алмазов. Не забывая про оформление отступов, прямо под инструкцией `while` игрового цикла добавьте следующие строки:

```
# игровой цикл
while not gameOver:
  createDiamonds(arenaPos, DIAMONDS[level])
  diamondsLeft = DIAMONDS[level]
```

   Здесь дополнительно создается переменная `diamondsLeft`. Она будет хранить количество алмазов, которые предстоит подобрать.

5. Сохраните и запустите программу. Так как вы сразу оказываетесь на первом уровне, в случайных местах на игровом поле должны появиться три алмаза.

После создания алмазов нужно добавить программный код, который будет следить за их сбором с помощью функции `pollBlockHits` (с ней вы познакомились в Приключении 4). Когда игрок подберет алмаз, функция присвоит блоку с типом `DIAMOND` тип `AIR`.

Добавьте следующий код в цикл прохождения уровня, который будет превращать блоки типа `DIAMOND_BLOCK` в блоки типа `AIR`:

1. Не забывая про отступы относительно начала цикла прохождения уровня, вызовите функцию `pollBlockHits`, чтобы получить все события выбора блоков:

```
  # цикл прохождения уровня
  while not gameOver and not levelComplete:
    hits = mc.events.pollBlockHits()
```

2. Обойдите в цикле все блоки, для которых произошел выбор, и определите их типы:

```
    for hit in hits:
      blockHitType = mc.getBlock(hit.pos.x,
                                 hit.pos.y,
                                 hit.pos.z)
```

3. Сравните тип очередного блока с типом `DIAMOND_BLOCK`. Если типы совпадают, присвойте блоку тип `AIR` и уменьшите на одну переменную `diamondsLeft`:

```
if blockHitType == block.DIAMOND_BLOCK.id:
  mc.setBlock(hit.pos.x,hit.pos.y, hit.pos.z,
              block.AIR.id)
  diamondsLeft = diamondsLeft - 1
```

   С помощью переменной `diamondsLeft` программа будет проверять, все ли алмазы собраны к моменту, когда персонаж достигнет дальнего края игрового поля.

4. Сохраните и запустите программу. Попробуйте подобрать алмаз — блок с типом `DIAMOND` — и убедитесь, что он исчезает, превратившись в блок типа `AIR`.

Не забывайте: чтобы подобрать блок, его нужно коснуться мечом и щелкнуть правой кнопкой мыши.

Можно ли усложнить сбор алмазов? Вероятно, вам удастся заставить алмазы двигаться вверх-вниз и разрешить игроку подбирать их, лишь когда они находятся в воздухе.

Ограничение по времени

Если дать игроку неограниченное время, игра станет по-настоящему простой (и скучной). Введя ограничение по времени, вы усложните игру, а заставив игрока собрать все алмазы и добраться до противоположной стороны игрового поля до окончания отведенного времени, вы дадите ему цель.

Если отведенное время будет превышено, игра должна закончиться. Ваша следующая задача — добавить инструкции в цикл прохождения уровня, которые будут проверять, не превышено ли отведенное время, и в случае данного нарушения присваивать флагу `gameOver` значение `True`:

1. Не забывая про отступы относительно начала цикла прохождения уровня, вычислите, сколько секунд осталось до конца интервала, отведенного для прохождения текущего уровня:

```
# цикл прохождения уровня
  while not gameOver and not levelComplete:
    secondsLeft = TIMEOUTS[level] - (time.time() - start)
```

2. Если количество оставшихся секунд меньше нуля, присвойте переменой `gameOver` значение `True` и выведите сообщение в чат:

```
    if secondsLeft < 0:
      gameOver = True
      mc.postToChat("Out of time...")
```

3. Сохраните и запустите программу. Через 30 секунд (так как это первый уровень) программа должна завершиться и вывести в чат сообщение «Out of time...» («Время истекло»), как показано на рис. 8.12.

Рис. 8.12. Сообщите игроку, когда отпущенное ему время закончится

УГЛУБЛЯЕМСЯ В КОД

Количество секунд, оставшихся от времени, отпущенного для прохождения уровня, вычисляется с помощью инструкции:

```
secondsLeft = TIMEOUTS[level] - (time.time() - start)
```

Значение для переменной `secondsLeft` вычисляется так. Количество секунд для прохождения текущего уровня берем из константы `TIMEOUTS`:

```
TIMEOUTS[level]
```

Из него вычитаем количество секунд, уже потраченных игроком на этом уровне, которое определяется как разность текущего времени и времени начала прохождения уровня:

```
(time.time() - start)
```

Наблюдение за положением персонажа

Когда игрок соберет все алмазы и приведет персонажа к противоположному краю игрового поля, он завершит прохождение уровня. Если персонаж упадет в ров или яму-ловушку, его

следует вернуть в начало игрового поля. Чтобы все это реализовать, программа должна знать, где находится персонаж.

После проверки местоположения персонажа программа может установить флаг `levelComplete` в значение `True`, если все алмазы собраны, или переместить персонажа в начало игрового поля.

Когда вы закончите этот раздел, в игру вполне уже можно будет играть, правда, на самом первом и простом уровне. Это может пригодиться вам, чтобы практиковаться в ее прохождении, потому что в следующем разделе вы усложните игру!

Ваша следующая задача — добавить в цикл прохождения уровня инструкции, которые будут отслеживать местоположение персонажа и возвращать его в начало или завершать уровень:

1. Не забывая про отступы относительно начала цикла прохождения уровня, получите позицию персонажа:

```
# цикл прохождения уровня
while not gameOver and not levelComplete:
  pos = mc.player.getTilePos()
```

2. Проверьте координату *y* персонажа. Если она ниже уровня игрового поля, персонаж провалился в ров или яму-ловушку, и его нужно вернуть в начало:

```
  if pos.y < arenaPos.y:
    mc.player.setPos(arenaPos.x + 1,
                     arenaPos.y + 1,
                     arenaPos.z + 1)
```

3. Проверьте, достиг ли персонаж конца игрового поля и все ли алмазы собрал. Это можно сделать, сравнив координату *z* персонажа с координатой *z* дальнего края игрового поля. Если координаты совпадают, присвойте флагу `levelComplete` значение `True`:

```
  if pos.z == arenaPos.z + ARENAZ and diamondsLeft == 0:
    levelComplete = True
```

   Когда флаг `levelComplete` получит значение `True`, цикл прохождения уровня завершится, на игровом поле будут созданы новые алмазы и игра начнется с самого начала.

4. Сохраните и запустите программу. Когда все алмазы будут собраны и персонаж окажется на дальнем краю игрового поля, игра должна запуститься сначала. Если персонаж провалится в ров или угодит в яму-ловушку, он должен автоматически вернуться в начало.

Завершение уровня и начисление очков

Когда игрок завершит уровень, программа должна вычислить заработанные им очки и перевести игру на следующий уровень. Очки начисляются по завершении уровня: за каждый

алмаз игрок получает одно очко. После суммирования очков за собранные алмазы они умножаются на число оставшихся секунд.

Для этого нужно добавить следующий программный код. Новые строки должны иметь по одному отступу как принадлежащие игровому циклу, но располагаться после цикла прохождения уровня:

1. Не забывая про отступы относительно начала игрового цикла, но после цикла прохождения уровня, проверьте, был ли пройден уровень:

```
# игровой цикл
while not gameOver:
  [прежний код]

  # цикл прохождения уровня
  while not gameOver and not levelComplete:
    [прежний код]

  if levelComplete:
```

2. Если уровень успешно пройден, вычислите сумму очков и прибавьте ее к переменной `points`, а затем выведите результат в чат:

```
    points = points + (DIAMONDS[level] * int(secondsLeft))
    mc.postToChat("Level Complete - Points = " + ↩
      str(points))
```

3. Переведите игру на следующий уровень, прибавив 1 к переменной `level`:

```
    level = level + 1
```

4. Если был пройден последний уровень, присвойте флагу `gameOver` значение `True` и выведите в чат сообщение с поздравлением:

```
    if level == LEVELS:
      gameOver = True
      mc.postToChat("Congratulations - All levels complete")
```

   Константа `LEVELS` хранит общее число уровней в игре.

5. Сохраните и запустите программу.

Ваша игра почти завершена! Если игрок соберет все алмазы и достигнет противоположного края игрового поля до момента, когда истечет отведенное ему время, игра перейдет на следующий уровень. Дальше она станет сложнее: игроку нужно собрать большее количество алмазов за меньшее время. Если игрок пройдет все уровни, игра закончится и в чат будет выведено сообщение с поздравлением.

Попробуйте добавить еще несколько уровней. Можно предоставить возможность начать игру в более медленном темпе, на первых уровнях дав больше времени и меньше алмазов, и плавно увеличивать сложность.

Добавление сообщения о завершении игры

Последнее, что нужно сделать, чтобы завершить разработку игры, — добавить в конце вывод сообщения о завершении игры и о количестве заработанных очков (рис. 8.13). Для этого добавьте следующую строку в самый конец программы:

```
mc.postToChat("Game Over - Points = " + str(points))
```

Рис. 8.13. Сообщите игроку, что игра закончена, и покажите количество заработанных очков

Все уровни пройдены, заработано 156 очков. Попробуйте побить этот рекорд!

Запишите количество очков в файл и создайте турнирную таблицу, а в конце игры показывайте игроку, какое место он занял.

Полный код программы «Коварные препятствия» можно загрузить на сайте книги www.wiley.com/go/adventuresinminecraft2e.

Краткая справочная таблица	
Команды	Описание
`import threading`	Импортирует модуль `threading`

`t = threading.Thread(`
`  target = function,`
`  args = (variable1, variable2))`
	Вызывает функцию в отдельном потоке выполнения

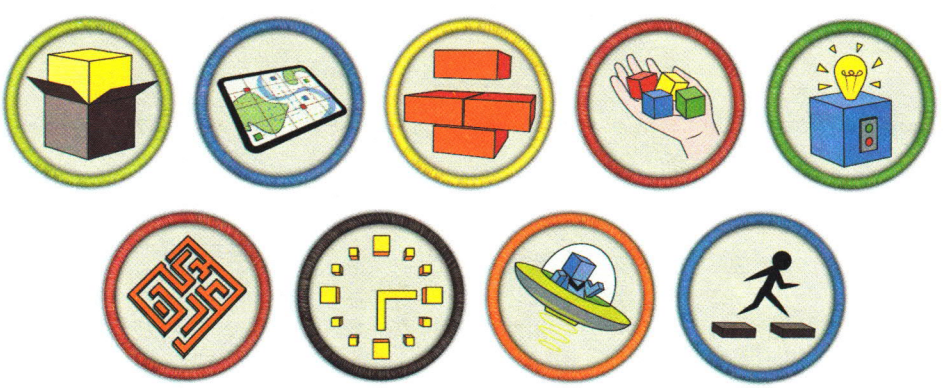
`t.start()`	Запускает поток и вызывает функцию
`t.join()`	Ожидает завершения потока
`import time`	Импортирует модуль `time`
`timeNow = time.time()`	Возвращает текущее время

Дополнительные приключения в путешествиях по миру Minecraft

Игра Minecraft дает фантастическую основу для творчества и приключений. Прибавьте к ней возможность управлять игрой с помощью программ, и единственным ограничением станет ваше воображение. Что делать дальше?

Ниже перечислены идеи и ресурсы, где вы можете почерпнуть вдохновение:

- Создание игр — отличный способ приобрести богатый опыт программирования. Загляните на сайт www.classicgamesarcade.com, где можно найти несколько интересных идей.
- В Minecraft могут играть сразу несколько игроков. Почему бы не воспользоваться этим и не написать программу, которой могли бы пользоваться многие люди?
- В интернете есть множество открытых источников данных. Попробуйте объединить Minecraft с веб-сайтами, такими как Twitter (dev.twitter.com), или добавить вывод прогноза погоды с сайта Met Office (www.metoffice.gov.uk/datapoint).



Новое достижение: создатель большого проекта для Minecraft!

Приложение A
Справочный материал

Таблица 1. Входные и выходные значения, переменные Python

Пометки с помощью комментариев

```
# целая строка с комментарием
print("hello") # конец строки
с комментарием
```

Использование констант

```
SIZE = 100 # имя верхнего регистра
```

Вывод на экран

```
print(10)      # вывести число
print("hello") # вывести строку
print(a)       # вывести переменную
```

Чтение с клавиатуры (Python V3)

```
name = input("your name?")
age  = int(input("how old?"))
```

Сохранение значений в переменных

```
a = 10   # целочисленное (целое) число
b = 20.2 # десятичное число
message = "hello" # строка
finished = True   # булево значение
high_score = None # без значения
```

Использование булевых переменных

```
anotherGo = True
while anotherGo:
  choice = int(input("choice?"))
  if choice == 8:
    anotherGo = False
```

Преобразование строк в (целочисленные) числа

```
size = int(size)
size = size + 1
```

Преобразование чисел в строки

```
age = str(age)
print("Your age is " + age)
```

Вычисление с помощью числовых переменных

```
b = a + 1  # сложение
c = a - 1  # вычитание
d = a * 2  # умножение
e = a / 2  # деление
f = a % 2  # остаток после деления
```

Вычисление абсолютных значений и разностей

```
absolute   = abs(-1) # абсолютное значение
1
difference = abs(x1-x2)
smallest   = min(x1, x2)
largest    = max(x1, x2)
```

Таблица 2. Циклы Python

Введение цикла

```
a = 0
while a<100:
  print(a)
  a = a + 1
```

Использование циклов со счетчиком

```
for a in range(10):
  print(a) # 0..9
for a in range(5, 50, 5):
  print(a) # 5..45 in steps of 5
```

Таблица 2. Циклы Python (продолжение)

Зацикливание всех символов в строке	Разбивка строк, разделенных запятой

`name = "Charlotte"`
`for ch in name:`
`  print(ch)`
	`line = "one,two,three"`
	`data = line.split(",")`
	`for d in data:`
	`  print(d)`

Таблица 3. Условные операторы Python

Использование инструкций if	Использование инструкций if/else

`if a==42:`
`print("the ultimate answer")`
	`if a==5:`
	`  print("correct")`
	`else:`
	`  print("wrong")`

Использование elif/else для многовариантного выбора	**Проверка многовариантных условий с *and, or***

`if a==1:`
`  print("option A")`
`elif a==2:`
`  print("option B")`
`else:`
`  print("anything else")`
	`if choice>0 and choice<=8:`
	`  print("in range")`
	`if choice<1 or choice>8:`
	`  print("out of range")`

Проверка *different* или *same* с помощью *if*	**Проверка *less* с *if***

`if a!=1:`
`  print("not equal")`
`if a==1:`
`  print("equal")`
	`if a<1:`
	`  print("less than")`
	`if a<=1:`
	`  print("less than or equal")`

Проверка противоположного значения с помощью *not*	**Проверка наибольшего значения с помощью *if***

`playing = True`
`if not playing:`
`  print("finished")`
	`if a>1:`
	`  print("greater than")`
	`if a>=1:`
	`  print("greater than or equal")`

Таблица 4. Функции Python

Определение и вызов функции	Передача параметров функции

`def myname():`
`  print("my name is Laura")`
`myname()`
`myname()`
	`def hello(name):`
	`  print("hello " + name)`
	`hello("Andrew")`
	`hello("Geraldine")`

Таблица 4. Функции Python

Возвращение значений из функции	Запись глобальных значений внутри функции

`def smallest(a, b):`
`  if a<b:`
`  return a`
`  return b`
`print(smallest(10, 20))`
	`treasure_x = 0`
	
	`def build():`
	`  global treasure_x`
	`  treasure_x = 10`

Таблица 5. Списки Python

Создание списка	Добавление в конец списка
`a = [] # пустой список` `a = [1, 2, 3]# заполненный список`	`a.append("hello")`
Вывод содержимого списка	**Задание размера списка**
`print(a)`	`print(len(a))`
Доступ к элементам в списке по их индексу	**Доступ к последнему элементу в списке**
`print(a[0]) # 0=первый элемент, 1=второй`	`print(a[-1])`
Удаление последнего элемента в списке	**Зацикливание всех элементов в списке**
`word = a.pop() # удалить последний элемент` `print(word)  # элемент удален`	`for name in ["David","Gail","Janet"]:` `  print(name)`

Таблица 6. Прочие модули Python

Задержка на малый промежуток времени	Генерирование случайных чисел
`import time` `time.sleep(1) # ждать одну секунду`	`import random` `print(random.randint(1,100))`
Получение текущей даты/времени	**Использование математических функций**

`import datetime`
`dateNow = datetime.datetime.now()`
`import time`
`timeNow = time.time()`
	`import math`
	`radians  = math.radians(angle)`
	`sin  = math.sin(radians)`
	`cos  = math.cos(radians)`
	`squareRoot = math.sqrt(number)`

Таблица 7. Обработка файлов

Чтение строк из файла	Запись строк в файл

`f = open("data.txt", "r")`
`tips = f.readlines()`
`f.close()`
`for t in tips:`
`  print(t)`
	`f = open("scores.txt", "w")`
	`f.write("Victoria:26000\n")`
	`f.write("Amanda:10000\n")`
	`f.write("Ria:32768\n")`
	`f.close()`

Получение списка совпадающих имен файлов	**Удаление нежелательных пробелов из строк**

`import glob`
`names = glob.glob("*.csv")`
`for n in names:`
`  print(n)`
	`a = "\n\n hello \n\n"`
	`a = a.strip()`
	`print(a)`

В табл. 8 показаны ключевые функции API Minecraft Python. Полный список функций можно найти по адресу www.stuffaboutcode.com/p/minecraft-api-reference.html.

Таблица 8. Minecraft API

Импорт Minecraft API	Импорт и использование констант имени блока
`import mcpi.minecraft as minecraft`	`import mcpi.block as block` `b = block.DIRT.id`
Подключение к игре Minecraft	**Вывод сообщения в чат Minecraft**
`mc = minecraft.Minecraft.create()`	`mc.postToChat("Hello Minecraft")`
Получение расположения персонажа	**Настройка расположения персонажа**

`# координаты плитки, на которой стоит`
`# персонаж?`
`pos = mc.player.getTilePos()`
`x = pos.x`
`y = pos.y`
`z = pos.z`
	`x = 5`
	`y = 3`
	`z = 7`
	`mc.player.setTilePos(x, y, z)`

Получение позиции персонажа	**Настройка позиции персонажа**

`# получить точную позицию персонажа`
`# в мире (e.g. x=2.33)`
`pos = mc.player.getPos()`
`x = pos.x`
`y = pos.y`
`z = pos.z`
	`# установить точную позицию персонажа`
	`x = 2.33`
	`y = 3.95`
	`z = 1.23`
	`mc.setPos(x, y, z)`

Получение высоты игрового мира	**Получение типа блока на позиции**
`# позиция Y первого блока не типа AIR` `height = mc.getHeight(5, 10)`	`# id блока (например, block.DIRT.id)` `blockId = mc.getBlock(10, 5, 2)`

Таблица 8. Minecraft API

Настройка/изменение блока на позиции	Настройка/изменение множества блоков за один раз
`mc.setBlock(5, 3, 2, block.DIRT.id)`	`mc.setBlocks(0,0,0, 5,5,5, block.AIR.id)`
Определение того, какие блоки были выбраны	**Очистка выбора любого блока**

`# какой блок был выбран в последний раз?`
`events = mc.events.pollBlockHits()`
`for e in events:`
`  pos = e.pos`
`  print(pos.x)`
	`# очистить(игнорировать) выбор с последнего`
	`# раза`
	`mc.events.clearAll()`

В табл. 9 показаны образцы типов блоков, доступных в Minecraft, которые можно использовать на ПК. Там, где это необходимо, включены значения данных, которые могут использоваться. Полный список идентификаторов блоков и значений блоков данных можно найти по адресу www.stuffaboutcode.com/p/minecraft-api-reference.html.

Таблица 9. Типы блоков Minecraft API

ID	Константа	ID данных	Подтип
0	AIR	-	-
1	STONE	-	-
2	GRASS	-	-
3	DIRT	-	-
4	COBBLESTONE	-	-
5	WOOD_PLANKS	0	Oak
		1	Spruce
		2	Birch
		3	Jungle
7	BEDROCK	-	-
8	WATER	-	-
9	WATER_STATIONARY	0	High
		..7	Low
10	LAVA	-	-
11	LAVA_STATIONARY	0	High
		..7	Low
12	SAND	-	-
13	GRAVEL	-	-
14	GOLD_ORE	-	-
15	IRON_ORE	-	-

Таблица 9. Типы блоков Minecraft API (продолжение)

16	COAL_ORE	-	-
17	WOOD	0	Oak (up/down)
		1	Spruce (up/down)
		2	Birch (up/down)
		3	Jungle (up/down)
		4	Oak (east/west)
		5	Spruce (east/west)
		6	Birch (east/west)
		7	Jungle (east/west)
		8	Oak (north/south)
		9	Spruce (north/south)
		10	Birch (north/south)
		11	Jungle (north/south)
		12	Oak (only bark)
		13	Spruce (only bark)
		14	Birch (only bark)
		15	Jungle (only bark)
18	LEAVES	1	Oak leaves
		2	Spruce leaves
		3	Birch leaves
20	GLASS	-	-
24	SANDSTONE	0	Sandstone
		1	Chiselled Sandstone
		2	Smooth Sandstone
35	WOOL	0	White
		1	Orange
		2	Magenta
		3	Light Blue
		4	Yellow
		5	Lime
		6	Pink
		7	Grey

Таблица 9. Типы блоков Minecraft API (продолжение)

		8	Light grey
		9	Cyan
		10	Purple
		11	Blue
		12	Brown
		13	Green
		14	Red
		15	Black
37	FLOWER_YELLOW	-	-
38	FLOWER_CYAN	-	-
41	GOLD_BLOCK	-	-
42	IRON_BLOCK	-	-
45	BRICK_BLOCK	-	-
46	TNT	0	Inactive
		1	Ready to explode
49	OBSIDIAN	-	-
50	TORCH	0	Standing on the floor
		1	Pointing east
		2	Pointing west
		3	Pointing south
		4	Pointing north
53	STAIRS_WOOD	0	Ascending east
		1	Ascending west
		2	Ascending south
		3	Ascending north
		4	Ascending east (upside down)
		5	Ascending west (upside down)
		6	Ascending south (upside down)
		7	Ascending north (upside down)
57	DIAMOND_BLOCK	-	-
80	SNOW_BLOCK	-	-
89	GLOWSTONE_BLOCK	-	-

Таблица 10. MinecraftStuff API (MinecraftDrawing)

Импорт MinecraftStuff API	**Создание объекта MinecraftDrawing**

`import mcpi.minecraftstuff as ↩ minecraftstuff`	`mc = minecraft.Minecraft.create()`
	`mcdrawing =`
	`  minecraftstuff.MinecraftDrawing(mc)`

Создание линии между двумя точками	**Получение всех позиций блока на линии в качестве списка**

`mcdrawing.drawLine(0, 0, 0,`
`  10, 10, 10, block.DIRT.id)`
	`line = mcdrawing.getLine(`
	`0,0,0,10,10,10)`
	`pos1 = line[0]`
	`print(pos1.x)`

Создание сферы	**Создание окружности**
`mcdrawing.drawSphere(0, 0, 0,` `  radius, block.DIRT.id)`	`mcdrawing.drawCircle(0, 0, 0,` `  radius, block.DIRT.id)`
Создание плоского многоугольника (например, треугольника)	

`tri = Points()`
`filled = True`
`tri.add(0,0,0)`
`tri.add(10,0,0)`
`tri.add(5,10,0)`
`mcdrawing.drawFace(tri, filled,`
`  block.DIRT.id)`	

Таблица 11. MinecraftStuff API (MinecraftShape))

Создание MinecraftShape	**Создание и стирание фигуры**

`mc = minecraft.Minecraft.create()`
`pos = mc.player.getTilePos()`
`shape =`
`  minecraftstuff.MinecraftShape(mc,`
`  pos)`
`shape.setBlock(0,0,0,block.DIRT.id)`
`shape.setBlock(1,0,0,block.DIRT.id)`
	`shape.draw()`
	`shape.clear()`

Перемещение фигуры на позицию	**Перемещение фигуры на несколько блоков**
`shape.move(10, 10, 10)`	`ymove = 1` `shape.moveBy(0, ymove, 0)`

ДЭВИД ВЭЙЛ,
МАРТИН О`ХЭНЛОН

PYTHON 3

MINECRAFT

ПРОГРАММИРУЙ СВОЙ МИР

Любишь играть в Minecraft?
Тебе нравится узнавать новое и придумывать то, чего раньше не существовало?
Хочешь построить собственный виртуальный мир,
которому будут завидовать все друзья?

**Можно ли объединить Minecraft и программирование? Нужно!
Теперь ты будешь не только играть и жить в удивительном мире Minecraft, но и научишься программировать на Python. Простые инструкции и советы помогут воплотить твои идеи в жизнь, построить дом и 3D-копировальную машину, найти сокровища и даже завести в своем «огороде» гигантские работающие часы.**

Прочитай эту книгу и превратись в настоящего демиурга, который способен создать свой мир и защитить его от инопланетян.

ПИТЕР®

Заказ книг:
тел.: (812) 703-73-74
books@piter.com

WWW.PITER.COM
каталог книг
и интернет-магазин

vk.com/**piterdetstvo**
instagram.com/**piterdetstvo**
facebook.com/**piterbooks**
youtube.com/**ThePiterBooks**

EAC

ISBN 978-5-4461-0951-7

Изготовлено в России. Изготовитель: ООО «Прогресс книга». Место нахождения и фактический адрес: 194044, Россия, г. Санкт-Петербург, Б. Сампсониевский пр., д. 29А, пом. 52. Тел.: +78127037373. Дата изготовления: 06.2018. Наименование: детская литература. Срок годности: не ограничен.